175 PTAS

87

mi computer

CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR



Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO
DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen VIII-Fascículo 87

Director:	José Mas Godayol
Director editorial:	Gerardo Romero
Jefe de redacción:	Pablo Parra
Coordinación editorial:	Jaime Mardones Francisco Martín
Asesor técnico:	Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Aribau, 185, 1.°, 08021 Barcelona
Tel. (93) 200 19 02

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

© 1983 Orbis Publishing Ltd., London
© 1984 Editorial Delta, S. A., Barcelona
ISBN: 84-85822-83-8 (fascículo) 84-7598-067-2 (tomo 7)
84-85822-82-X (obra completa)
Depósito Legal: B. 52-84

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 258509
Impreso en España-Printed in Spain-Septiembre 1985

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 19 425 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 429 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S. A. (Aribau, 185, 1.°, 08021 Barcelona), o también con talón bancario remitido a la misma dirección.

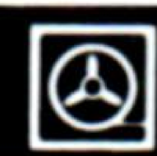
c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

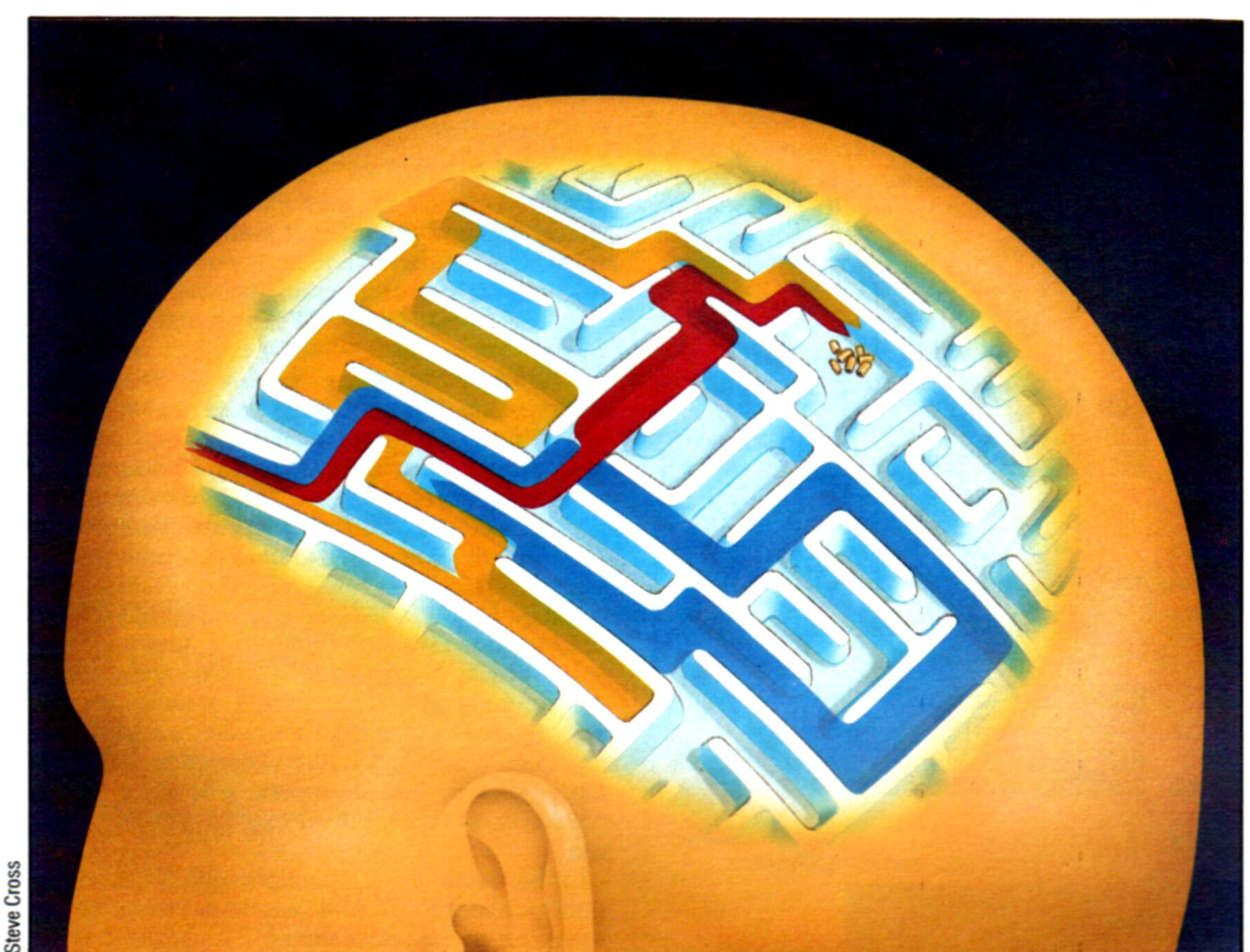
Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

Para cualquier aclaración, telefonear al (93) 215 75 21.

No se efectúan envíos contra reembolso.

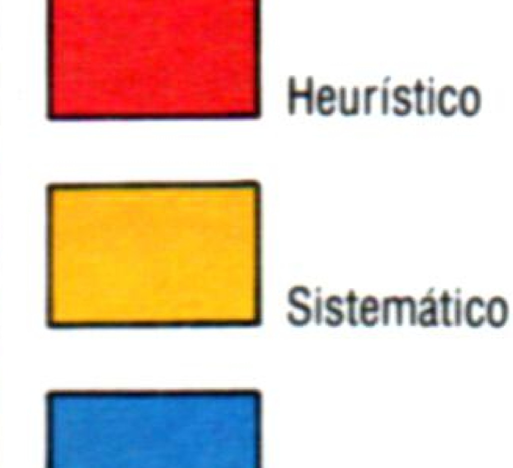






Steve Cross

La carrera de las ratas
El diagrama ilustra tres rutas posibles a través del laberinto utilizando tres estrategias de búsqueda diferentes: aleatoria, sistemática y heurística. En este ejemplo, el método heurístico encuentra el objetivo antes que los otros dos métodos, si bien sería posible diseñar un laberinto en el cual no fuera así. En la práctica, a menudo los laberintos se resuelven mejor mediante una combinación de métodos de búsqueda heurístico y exhaustivo (en el cual se consideran todas las posibilidades)



Búsqueda inteligente

El desarrollo de técnicas de búsqueda constituye un importante área en la investigación de la inteligencia artificial

Imaginemos que se sueltan tres ratas en un laberinto en cuyo interior hay, en algún lugar, un cuenco lleno de apetitosas bolitas para ratas. Una de ellas deambula por el laberinto durante algunos minutos y después se queda dormida (el insensible experimentador ha estado echando ginebra en el agua de la que bebía el animal). La segunda rata es más metódica: coloca su pata delantera izquierda contra un pared y prosigue efectuando giros a la izquierda ante cada bifurcación, retrocediendo cuando detecta una calle sin salida. Finalmente alcanza su objetivo, pero para entonces la tercera rata ya se ha comido todo el alimento.

La tercera rata posee un delicado sentido del olfato. En varios puntos durante la búsqueda se detiene a olfatear el aire y sigue el camino que, según percibe, la conducirá más cerca del delicioso aroma. Es posible idear un laberinto que despiste a esta rata (así como es posible idear uno que confunda a la seguidora de paredes), pero casi todos estaríamos de acuerdo en que este tipo de estrategia de búsqueda es la más inteligente de las tres.

El concepto de búsqueda es clave en el campo de la inteligencia artificial. Tanto si se halla a 70 m de profundidad en las aguas del Caribe con una escafandra autónoma de submarinista buscando un tesoro sumergido, como sentado a una mesa, concentrado en la resolución de un crucigrama, usted va en busca de algo. El solucionador de problemas de ecuaciones metafóricas mediante búsqueda ha demostrado ser muy provechoso en la AI, porque se pueden tratar muchos problemas diferentes mediante el empleo de técnicas de búsqueda. Nuestras tres ratas ilustran diferentes clases de estrategia:

- Búsqueda aleatoria (el "recorrido de borracho")
- Búsqueda exhaustiva (enumeración sistemática)
- Búsqueda heurística (exploración guiada)

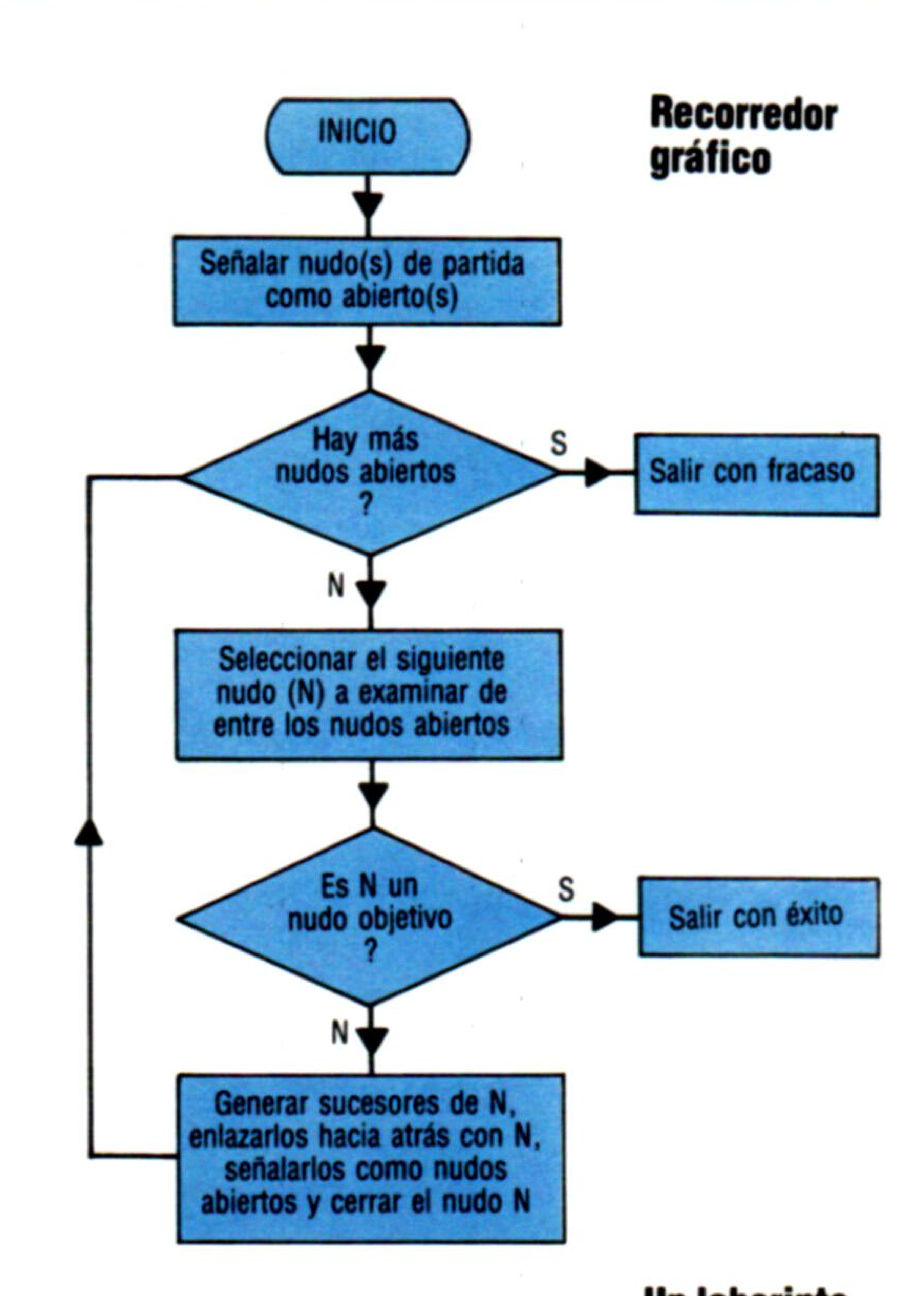
Se considera que el tercero es un enfoque más inteligente al problema que los otros dos, porque normalmente requiere menos esfuerzo para llegar a la solución. Pero todos los métodos heurísticos dependen de alguna forma de saber cuándo la búsqueda se está aproximando a su objetivo (el sentido del olfato de la rata, p. ej.). Sin conocimientos no se puede ser inteligente: uno tiene que depender de la enumeración sistemática.

Hallar una ruta a través de un laberinto es un problema de búsqueda típico. (Sin embargo, es importante comprender que muchos problemas que no son espaciales, como la integración de una expresión matemática simbólica, se pueden tratar uti-

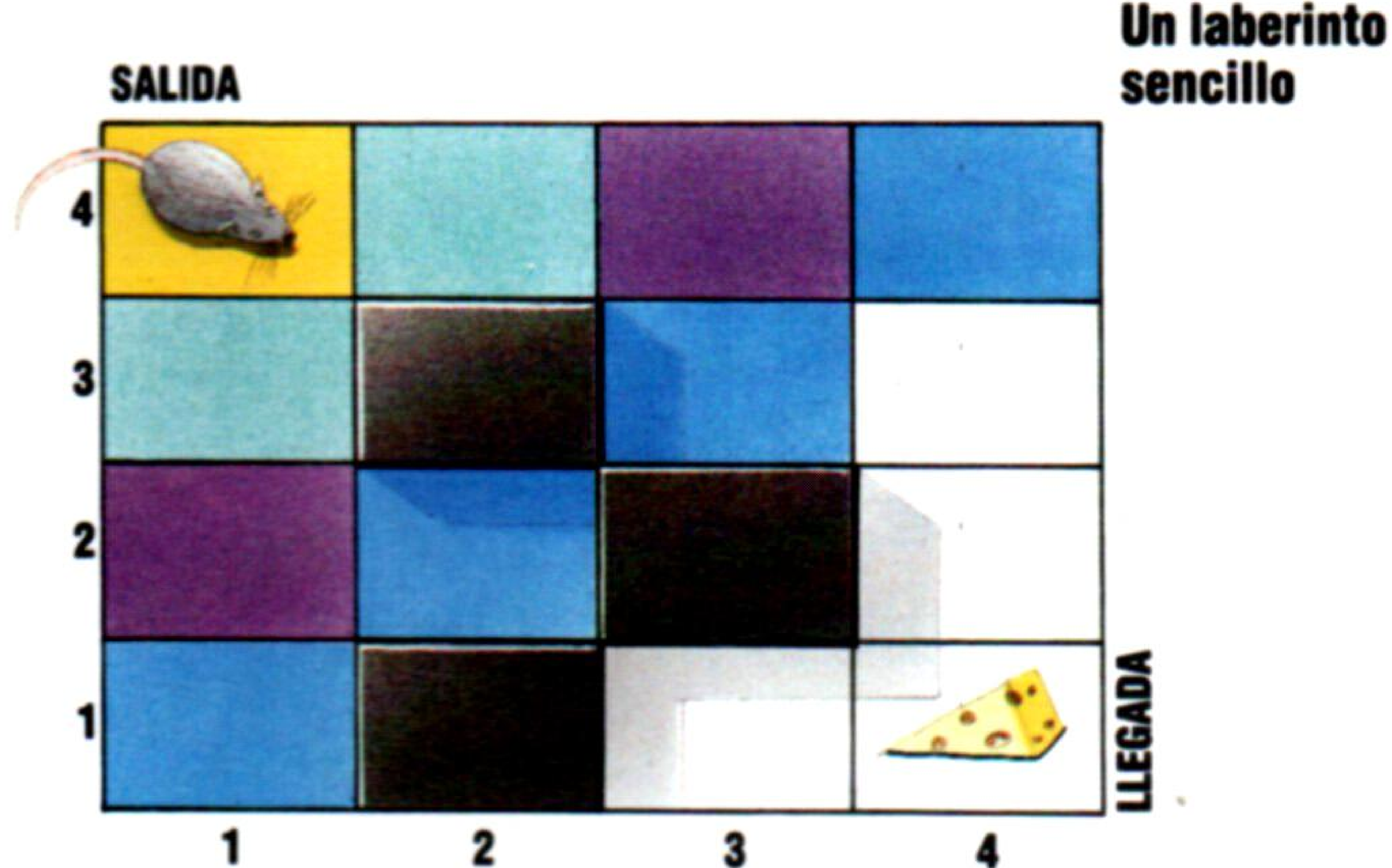
Cuestiones de búsqueda

La mayoría de los métodos de búsqueda de AI implican la generación de árboles a partir del trazado del laberinto a recorrer. El árbol que vemos aquí se configuró a partir del laberinto tomando el punto de partida como el nudo raíz, y produciendo sucesivas generaciones mediante la consideración de todos los movimientos posibles desde el nudo en cuestión. Observe que no se permiten movimientos en diagonal. El diagrama de flujo esboza el algoritmo básico para generar la estructura arborescente

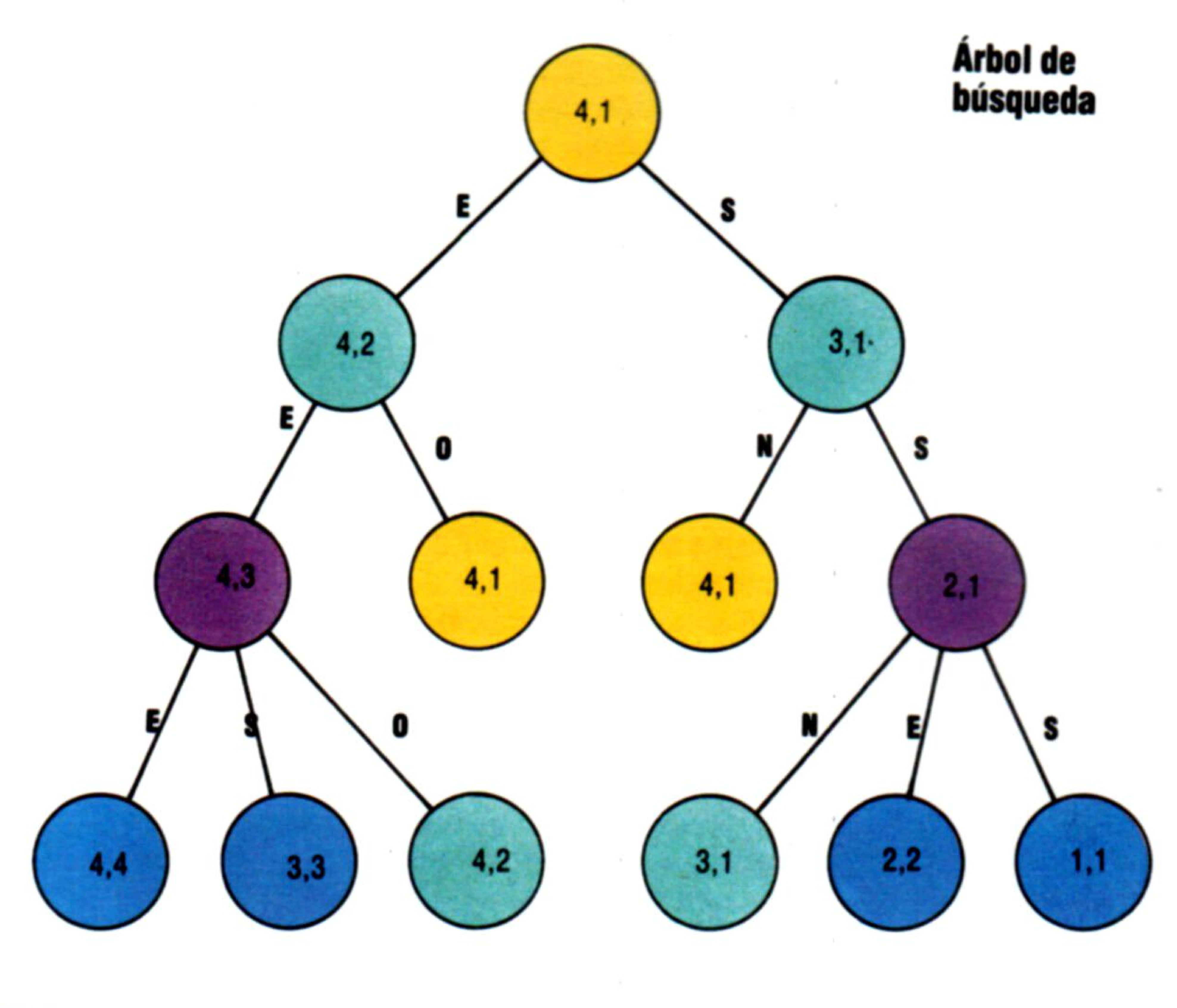
Recorredor gráfico



Un laberinto sencillo



Árbol de búsqueda



Kevin Jones

lizando el mismo marco de búsqueda.) Existen dos formas de juzgar la calidad de un método de búsqueda:

- ¿Cuánto tiempo se necesita para hallar una ruta?
- ¿Es la hallada la ruta más económica?

Lo ideal es contar con un método que encuentre la solución óptima en el menor tiempo posible. En la práctica, no obstante, podemos vernos obligados a optar ya sea por aceptar una solución rápida por debajo del nivel óptimo o bien tomarnos mucho tiempo para la mejor solución posible.

La mayor parte de las estrategias de búsqueda en inteligencia artificial se crean según un plan común, que permite dar mayor o menor énfasis a uno u otro de estos criterios de actuación. El diagrama de algoritmos de recorrido que vemos esboza una familia de métodos de búsqueda a partir de los cuales se puede seleccionar un miembro determinado mediante la elección de cómo rellenar el casillero central, o de cómo decidir qué nudo examinar a continuación. Los términos "abierto" y "cerrado" aluden al estado de cualquier nudo del camino desde el principio al punto de destino. Los nudos abiertos necesitan ser examinados; los nudos cerrados ya se han examinado. Los nudos abiertos se hallan en una especie de lista de espera, y la clave para la eficacia es el orden por el cual se procesará esta lista.

La búsqueda avanza configurando un árbol. El sencillo laberinto de 4×4 (izquierda) muestra a la rata-robot en la casilla superior izquierda y el alimento en la inferior derecha. En cada casilla la rata tiene una opción de hasta cuatro movimientos (norte, sur, este u oeste), si bien algunos pueden estar bloqueados. Si dibujamos las opciones abiertas a la rata al principio, podrá ver la estructura arborescente del proceso de búsqueda.

Desde el cuadrado inicial de (4,1), la rata sólo puede avanzar hasta el este a (4,2) o hacia el sur a (3,1). Desde (4,2) puede avanzar hasta el este o el oeste y desde (3,1) puede ir hacia el norte o hacia el sur. Si el roedor se mueve hacia el este y luego hacia el oeste (o al sur y después al norte) volverá a llegar al punto de partida, lo que no es muy inteligente pero refleja cómo la misma casilla puede aparecer en diferentes nudos del árbol, correspondiendo a las diversas formas de llegar a ellos.

Un método sistemático para explorar el árbol es el llamado *búsqueda por niveles*. Investiga los nudos por orden de proximidad al punto de partida o raíz. Por tanto, considera cada secuencia de N movimientos (a nivel N en el árbol) antes de cualquier secuencia de N+1 movimientos. Está destinado a hallar una ruta que exija la menor cantidad posible de pasos y, dado que cada paso es igualmente costoso, resultará ser la solución de mínimo costo. Pero hallarla podría llevar mucho tiempo. A medida que el árbol se vuelve más ancho y más profundo, el tiempo requerido para hallar el objetivo aumenta de forma exponencial.

Para mejorar una búsqueda por niveles se requiere una fuente de información heurística acerca de la distancia a la que se halla el objetivo. Como dicha medida podríamos utilizar la *Manhattan distance* (distancia de Manhattan). En la isla de Manhattan (estado de Nueva York), la mayoría de las calles se intersectan en ángulo recto. Para llegar de A a B uno debe avanzar tantas manzanas hacia el

norte o el sur y tantas otras hacia el este o el oeste. Del mismo modo, en el laberinto, una rata inteligente puede calcular a cuántas casillas de distancia se halla el objetivo.

Si se sabe que una búsqueda se está aproximando a la solución, puede ser acelerada. La rata de nuestro ejemplo inicial seguía una regla sencilla: acercarse cada vez más al objetivo. Ésta es la estrategia de *escalar montañas*, así llamada porque se la puede comparar con hallar la cima de una montaña cuando uno está desorientado avanzando siempre hacia arriba. Puede ser mucho más rápida que una búsqueda por niveles, pero no ofrece garantías de hallar la ruta óptima. Podría quedarse varada en un pico local (porque la búsqueda está orientada hacia *cualquier* cima).

Si se realiza una síntesis entre la estrategia de búsqueda por niveles y la de escalar montañas surge un método mejor: el llamado *algoritmo A-Star* (A*). Éste selecciona el nudo a examinar a continuación sobre la base de: HD+SF, donde HD es el cálculo heurístico de la distancia que queda y SF es la distancia cubierta hasta el momento. Cuanto mejor sea la estimación de HD, más eficaz es la búsqueda. Si el HD falla, estará evaluado en menos en vez de en más.

En nuestro programa de ejemplo, utilizamos una estructura de programa para implementar estos tres métodos, sólo con modificaciones menores. Las únicas diferencias se hacen evidentes en la elección del nudo a examinar a continuación:

Búsqueda por niveles	— tomar SF más bajo
Escalar montañas	— tomar HD más bajo
Algoritmo A*	— tomar HD+SF más bajo

Programa de búsqueda para el laberinto

```
1000 REM ****************************************
1010 REM ** Listado 2.1:                       **
1020 REM **      PROGRAMA BUSQUEDA LABERINTO    **
1030 REM ****************************************
1040 MODE7
1050 MH=17:MW=25:REM Altura y anchura del laberinto
1060 SI=256 : REM limites del arbol.
1070 WA=1: RR=2 :FO=3 :DN=4 :BL=5
1080 W1=1: REM peso de SF
1090 W2=2: REM peso de HD
1100 DIM M(MH+1,MW+1):REM el laberinto
1110 DIM C$(5) : REM caracteres del laberinto
1120 DIM P(SI),S(SI),N(SI),H(SI)
1130 REM Camino, Pasos,Nudo,Distancia-heuristica.
1140
1150 REM — Rata en el laberinto:
1160 GOSUB 1360 : REM hacer el laberinto
1170 NC=0 : REM n. de nudos examinados
1180 K=0 : REM contador
1190 GOSUB 1660 : REM borrar todos los caminos
1200 N(1)=2*MW+2 : REM 1er. nudo abierto
1210 S(1)=0: H(1)=FR-1+(FC-1)
1220 P(1)=0: REM ningun predecesor
1230 REM — Bucle principal:
1240 REM **** BUCLE PRINCIPAL ****
1250 GOSUB 1770 : REM tomar siguiente nudo S
1260 NC=NC+1
1270 PRINT TAB (0,22);NC,SR,SC,H(S);"
1280 GOSUB 1880 : REM generar sucesores
1290 IF (FR<>SR OR FC<>SC) AND NC<300 THEN 1240
1300 PRINT TAB (0,22);"Concluida la busqueda!"
1310 IF FR=SR AND FC=SC THEN GOSUB 2290 : REM rastrear pasos
1320 IF NC>300 THEN PRINT "Fracaso!"
1330 PRINT NC;"  nudos examinados."
1340 END
1350 :
1360 REM — Rutina para crear laberinto:
1370 FOR P=1 TO MH+1
1380 FOR R=1 TO MW+1
1390 IF RND(1)<0.28 THEN M(P,R)=WA ELSE M(P,R)=BL
1400 IF P=3 OR R=3 THEN M(P,R)=BL
1410 IF P=1 OR R=1 THEN M(P,R)=WA
1420 IF P>MH OR R>MW THEN M(P,R)=WA
1430 NEXT:NEXT
1440 FR=2+INT(RND(1)*(MH-1)):REM fila alimento
1450 FC=4+INT(RND(1)*MW-3): REM columna alimento
1460 M(FR,FC)=FO
1470 M(2,2)=RR : REM rata robot
1480 C$(BL)=" "
1490 C$(WA)=CHR$255 : REM gotita
1500 C$(DN)=" "
1510 C$(RR):="R"
1520 C$(FO)="f"
1530 GOSUB 1560 : REM visualizarlo
1540 RETURN
1550 :
1560 REM — Rutina visualizacion del laberinto:
1570 CLS
1580 FOR R=1 TO MH+1
1590 FOR C=1 TO MW+1
1600 PRINT TAB(C,R);C$(M(R,C));
1610 NEXT
1620 PRINT
1630 NEXT
1640 RETURN
1650 :
1660 REM — Rutina borrar arbol:
1670 DD=9999 : REM muerto
1680 FOR Q=1 TO SI
1690 P(Q)=0
1700 S(Q)=DD
1710 N(Q)=0
1720 H(Q)=DD
1730 NEXT
1740 NN=2 : REM siguiente nudo libre
1750 RETURN
1760 :
1770 REM — Tomar mejor nudo S:
1780 S=1: BN=DD
1790 FOR I=1 TO SI
1800 V=S(I)*W1+ABS(H(I))*W2
1810 IF V<BN AND H(I)>=0 THEN S=I:BN=V
1820 NEXT
1830 IF S=1 THEN PRINT TAB(0,20);"Explorando...."
1840 SR=INT(N(S)/MW)
1850 SC=N(S)-MW*SR
1860 RETURN
1870 :
1880 REM — Rutina para generar sucesores:
1890 IF H(S)=0 THEN RETURN: REM hecho.
1900 REM —Norte:
1910 Y=SR-1 : X=SC
1920 IF Y>1 THEN GOSUB 2090
1930 REM —Este:
1940 Y=SR : X=SC+1
1950 IF X<= MW THEN GOSUB 2090
1960 REM —Sur:
1970 Y=SR+1 : X=SC
1980 IF Y<= MH THEN GOSUB 2090
1990 REM —Oeste:
2000 Y=SR : X=SC-1
2010 IF X>1 THEN GOSUB 2090
2020 REM — tambien cerrar nudo S:
2030 H(S)=-H(S)
2040 IF H(S)>0 THEN PRINT "Ugh!"
2050 PRINT TAB(SC,SR);"  ";
2060 M(SR,SC)=DN
2070 REM celdas vacias en la pantalla.
2080 :
2090 REM — Rutina para abrir 1 nudo:
2100 IF M(Y,X)=DN THEN RETURN
2110 IF M(Y,X)=WA THEN RETURN
2120 REM — primero hallar posicion libre:
2130 NX=0
2140 REM ** BUCLE HALLAR POSICION
2150 IF S(NN)<>DD THEN NX=NX+1:NN=NN+1
2160 IF NN>SI THEN NN=1
2170 IF NX>SI THEN PRINT "Completo!":STOP
2180 IF S(NN)<>DD THEN 2140
2190 REM — Ahora abrirlo:
2200 XY=X+Y*MW
2210 N(NN)=XY
2220 P(NN)=S
2230 S(NN)=S(S)+1
2240 H(NN)=ABS(Y-FR)+ABS(X-FC)
2250 PRINT TAB(X,Y);"+";
2260 REM lo muestra en pantalla
2270 RETURN
2280 :
2290 REM — Rutina para desandar el camino:
2300 ST=S(S)
2310 FOR Q=1 TO 10000:NEXT Q:GOSUB 1560
2320 PRINT TAB(FC,FR);C$(FO);
2330 REM ** IMPRIMIR CAMINO **
2340 S=P(S) : REM nudo padre
2350 XY=N(S) : REM coords.
2360 Y=INT(XY/MW)
2370 X=XY-Y*MW
2380 M(Y,X)=RR : REM huella de la rata!
2390 PRINT TAB(X,Y);"*".
2400 IF S>0 THEN 2330
2410 PRINT TAB(2,2);C$(RR)
2420 PRINT TAB(0,22);"Camino de  ";ST;"  pasos."
2430 PRINT NC;"  nudos cerrados."
2440 RETURN
```

Líneas laberínticas
El laberinto está contenido en una matriz bidimensional, M(,), y las estructuras de datos para el árbol de búsqueda están retenidas en las matrices P(), S(), N() y H(). El programa combina factores de costo en curso y costo estimado hasta el objetivo, que se pueden regular alterando los valores W1 y W2 en las líneas 1080 y 1090. Tal como están listados los valores, el programa seguirá una estrategia de escalar montañas, pero quizá usted quiera experimentar. El método heurístico utilizado en este programa se basa en la *Manhattan distance*, que es de gran precisión en este ejemplo. Por consiguiente, inclinando la balanza hacia el método heurístico, haciendo que W2>W1, la búsqueda se acelerará

Complementos al BASIC

El programa está escrito para el BBC Micro. Para el C64 y el Spectrum, introducir estos cambios:

Commodore 64:
Reemplazar todos los PRINT TAB(X,Y);"MENSAJE", por P=X:Q=Y: GOSUB 3000

```
3000 :PRINT"MENSAJE"
3000 REM ** RUTINA TAB **
3010 PRINT CHR$(19);
3020 PRINT LEFT$(DW$,Q);
     TAB(P);
3030 RETURN

1040 FOR I=1 TO 25:
     DW$=DW$+CHR$(17):
     NEXT I
1490 C$(WA)=CHR$(102)
1570 PRINT CHR$(147)
```

Spectrum:
Sustituir todos los

TAB(X,Y); por AT X,Y;

```
1490 LET C$(WA)=CHR$(143)
```

Suprimir la línea 1040

	(1)	(2)	(3)	
T$()	Perlas	Figurillas	Especias	Descripciones de las mercancías que ofrece el jefe.
V1()	2 p.o.	2 p.o.	1 p.o.	Precios de mercancías al zarpar el barco.
V2()	?	?	?	Precios de mercancías al regresar el barco.
EQ(,) 1 saco de sal	0.5	0.5	1	Proporciones de trueque de las mercancías a intercambiar.
1 bala de tela	5	5	10	
1 joya	3	3	6	
1 cuchillo	2	2	4	

El Nuevo Mundo

Al desembarcar en el nuevo continente se producirá nuestro primer encuentro con los nativos

Al alcanzar la etapa final del juego, el viaje propiamente dicho concluye cuando el jugador consigue desembarcar en el Nuevo Mundo y comerciar con las mercancías que había adquirido al comenzar el juego. Cabe esperar que las mismas le reporten beneficios cuando regrese a puerto y venda las mercancías recientemente adquiridas. Pero el peligro no ha pasado aún: todavía hay que conocer a los habitantes de estas nuevas tierras, que pueden ser amistosos u hostiles. Por lo tanto, debe cuidarse el impedir cualquier acción que pueda provocar antagonismo entre ellos y su tripulación.

Cuando termina el bucle principal del viaje, la línea 891 llama a la subrutina 10000, que se ocupa de su llegada al Nuevo Mundo. A medida que el barco se acerca a la costa, varios grupos de nativos armados se hacen a la mar en canoas para investigar. Usted debe decidir ahora si corre el riesgo de que el barco sea atacado, negándose a disparar, o bien si abre fuego y desencadena, tal vez, una guerra.

La línea 10015 comprueba el segundo elemento de la matriz de suministros, OA(2), que indica la cantidad de armas que hay a bordo. De haber armas, debe decidir el curso de acción a seguir. La línea 10022 espera una respuesta; si digita un "sí" y emplea sus armas, morirán muchos de los nativos. Sin embargo, es probable que tal curso de acción impulse a los supervivientes a regresar al barco durante la noche y prenderle fuego, con lo cual terminaría el juego. Es obvio que no tiene sentido una agresión sin que medie provocación alguna. (Se ha insertado la línea 10044 para impedir la continuación del juego tras esta imprudente decisión.)

Si decide no disparar no habrá, por supuesto, ninguna necesidad de que los nativos devuelvan el golpe, y la línea 10026 enviará el programa a la línea 10050. En esta sección, los nativos abordan la nave de forma pacífica y dan la bienvenida a los viajeros, quienes son llevados al poblado, donde los recibe el jefe. Éste resulta ser bastante amistoso y en anteriores ocasiones ya ha realizado trueques con otros visitantes provenientes del Viejo Mundo. Después de comer y descansar, el intercambio comercial puede comenzar al día siguiente (de ello se ocupa un módulo separado). Pero, antes de que pueda iniciarse el intercambio, hemos de hacer algunos arreglos previos.

Las matrices de intercambio

Los precios de las perlas, las especias y las figurillas se determinaron cuando el barco abandonó el puerto, pero desde entonces han experimentado modificaciones.

Para facilitar el intercambio comercial, se crean varias matrices nuevas. La línea 60 Dimensiona la matriz T$(), que contiene descripciones de los tres tipos de mercancías que ofrece el jefe: perlas, figurillas y especias. La línea 61 DIMensiona una matriz, V1(), que contiene los precios de mercado de estas mercancías cuando el barco se hizo a la mar.

Las perlas y las figurillas se estaban vendiendo a dos piezas de oro cada una, y se podían comprar especias a una pieza de oro el gramo. La línea 62 DIMensiona la matriz V2(), que contiene los precios de las mercancías cuando el barco regrese a puerto. Pero como los valores de mercado de las mercancías fluctúan, un elemento aleatorio incide en la estipulación del precio final. El precio de V2(1), que representa las perlas, se establece en dos o 2 ½ piezas de oro mediante la expresión de la línea 62. En la misma línea, expresiones similares fijan el precio de las figurillas en una, dos o tres piezas de oro cada una; las especias valdrán dos o 2 ½ piezas el gramo.

Cuando se produce el intercambio comercial, las mercancías no se compran ni se venden por oro, sino que se realizan trueques según un valor de intercambio acordado entre las dos partes: un lote de mercancías a cambio de otro. El jefe ofrece al capitán del barco una cantidad de perlas, figurillas y especias por cada lote de mercancías que le ofrece el capitán. Por razones de simplicidad, daremos por sentado que los comerciantes del Nuevo Mundo poseen cantidades ilimitadas de las mercancías con que comerciarán. La cantidad de cada una se determina en función de las cantidades de cada mercancía que haya en el barco.

En las líneas 64-68 se DIMensiona una matriz bidimensional a la que se asignan valores, con las proporciones de trueque para cada producto. El primer subíndice corresponde a los cuatro artículos con los que usted comerciará: sal, tela, cuchillos y joyas. El segundo subíndice representa los tres artículos con los que comerciarán los nativos: perlas, figurillas y especias, especificándose las proporciones del trueque en tres líneas del programa.

La línea 64 establece las proporciones de trueque para sacos de sal, el primer elemento del primer subíndice. Las perlas son el primer elemento del segundo subíndice y a EQ(1,1) se le asigna un valor que da la proporción de perlas para sacos de sal. El establecimiento de EQ(1,1) en 0.5 fija la proporción de trueque en media perla por cada saco de sal. Del mismo modo, a EQ(1,2) se le asigna un valor que corresponde a la intersección del primer elemento del primer subíndice con el segundo elemento del segundo subíndice: sal y figurillas. Cuando EQ(1,2) se establece en 0.5, la proporción de trueque es una figurilla por cada dos sacos de sal. EQ(1,3)=1 establece la proporción de un gramo de especias, el tercer elemento del segundo subíndice, por cada saco de sal.

Las proporciones de trueque para el segundo elemento del primer subíndice (balas de tela) se determina en la línea 66; una bala vale 5 perlas, 5 figurillas o 10 gramos de especias. La línea 67 se ocupa del tercer elemento del primer subíndice, cuchillos, cada uno de los cuales vale tres perlas, tres figurillas o seis gramos de especias. La línea 68 se ocupa de las joyas, el cuarto elemento, y se pueden intercambiar por dos perlas, dos figurillas o cuatro gramos de especias.

En la línea 69 se DIMensiona una matriz, AO(3), para almacenar las cantidades de perlas, figurillas y especias adquiridas durante el intercambio. Ésta no debe confundirse con la matriz OA, que almacenaba la cantidad de cada mercancía adquirida al comienzo del viaje.

Módulo 11: La llegada

Dimensionamiento de las matrices del intercambio

```
60 DIM T$(3):T$(1)="PERLAS":T$(2)="FIGURILLAS":T$(3)="ESPECIAS"
61 DIM V1(3):V1(1)=2:V1(2)=2:V1(3)=1
62 DIM V2(3):V2(1)=2+(INT(RND(1)*1)/2):V2(2)=2+(INT(RND(1)*3)-1)
63 V2(3)=2+(INT(RND(1)*1)/2)
64 DIM EQ(4,3)
65 EQ(1,1)=0.5:EQ(1,2)=0.5:EQ(1,3)=1
66 EQ(2,1)=5:EQ(2,2)=5:EQ(2,3)=10
67 EQ(3,1)=3:EQ(3,2)=3:EQ(3,3)=6
68 EQ(4,1)=2:EQ(4,2)=2:EQ(4,3)=4
69 DIM AO(3)
```

Adición al cuerpo principal del programa

```
890 REM LLEGADA AL NUEVO MUNDO
891 GOSUB 10000
```

Subrutina de la llegada

```
10000 REM LLEGADA AL NUEVO MUNDO
10001 PRINTCHR$(147):GOSUB 9200
10005 S$="LLEGAS AL NUEVO MUNDO*":GOSUB 9100
10006 PRINT:GOSUB 9200
10007 S$="MIENTRAS TE APROXIMAS A LA COSTA*":GOSUB 9100
10009 S$="SALEN NATIVOS EN CANOAS PARA RECIBIRTE*":GOSUB 9100
10010 PRINT:GOSUB 9200
10015 IF OA(2)=0 THEN 10050
10017 S$="SU ASPECTO ES FIERO Y ESTAN ARMADOS!!*":GOSUB 9100
10018 PRINT:GOSUB 9200
10020 S$="ABRES FUEGO? (S/N)*":GOSUB 9100
10022 INPUT I$: I$=LEFT$(I$,1)
10024 IF I$<>"N" AND I$<>"S" THEN 10022
10026 IF I$="N" THEN 10050
10028 PRINT:GOSUB 9200
10030 S$="HAN MUERTO MUCHOS NATIVOS*":GOSUB 9100
10032 S$="PERO DURANTE LA NOCHE*":GOSUB 9100
10034 S$="OTROS REGRESAN*":GOSUB 9100
10036 S$="Y LE PRENDEN FUEGO A TU BARCO!!!!*":GOSUB 9100
10038 PRINT:PRINT:GOSUB 9200
10040 S$="          JUEGO TERMINADO*":GOSUB 9100
10042 END
10044 GOTO 10042
10050 S$="TE LLEVAN A CONOCER A SU*":GOSUB 9100
10052 S$="JEFE. YA HA CONOCIDO ANTES A GENTE DE*":GOSUB 9100
10054 S$="TU RAZA Y SE MUESTRA MUY AMABLE.*":GOSUB 9100
10056 GOSUB 9200
10058 S$="LA TRIPULACION HA COMIDO Y ESTA DESCANSANDO*":GOSUB
      9100
10060 PRINT:GOSUB 9200
10062 S$="MAÑANA COMENZARA LA ACTIVIDAD COMERCIAL*":GOSUB 9100
10064 PRINT:GOSUB 9200
10066 S$=K$:GOSUB 9100
10068 GET I$:IF I$="" THEN 10068
10069 RETURN
```

Complementos al BASIC

Spectrum:

Sustituir V1() por B(), V2() por D(), EQ(,) por Q(,) y AO() por E() en todo el listado e introducir las siguientes modificaciones:

```
60    DIM T$(3,9)
10001 CLS:GO SUB 9200
10022 INPUT I$:LET I$=I$(TO 1)
10068 LET I$=INKEY$:IF I$=""THEN GO TO
      10068
```

BBC Micro:

Introducir las siguientes modificaciones:

```
10001 CLS:GOSUB 9200
10068 I$=GET$
```

Ian McKinnell

GOTO END

En este último capítulo destacaremos la necesidad de describir adecuadamente los datos en PASCAL

El diseñador del PASCAL, Niklaus Wirth, tituló uno de sus libros *Data structures+algorithms=programs* (Estructuras de datos + algoritmos = programas), que refleja la importancia que posee la descripción de los datos para formular los algoritmos que procesa esa información. Si utilizamos una matriz de chars para representar una serie de caracteres, por ejemplo, las funciones y procedimientos necesarios para procesarlos (hallar su longitud, concatenarlos, etcétera) serán sumamente diferentes de los que se requerirían si se decidiera emplear en su lugar una lista enlazada.

Tomemos a modo de ejemplo la sencilla tarea de hallar la longitud de una serie, siendo la misma desconocida. Recuerde que hemos llenado todos los elementos de matriz "libres" de la representación de matriz con caracteres ASCII NUL, chr(0), o terminado la lista dinámica con el valor de puntero NIL. La versión de matriz parece algo simple:

```
FUNCTION Longitud (S:serie):Cardinal;
  VAR
    N           :0..LongitudSerie;
    hallada     :boolean;
  BEGIN
    N:=0;
    hallada:=false;
    REPEAT
      N:=N+1;
      hallada:=S[N]=chr(0)
    UNTIL hallada OR (N=LongitudSerie);
    IF hallada
      THEN
        Longitud:=N-1
      ELSE
        Longitud:=LongitudSerie
  END; {Longitud}
```

Programa Árbol Clasificador

Este programa utiliza algunos procedimientos del anterior programa ListaCirc, siendo éstos llamados en el momento adecuado. Los datos de nombres, cantidades adeudadas y cualquier otro campo que quiera añadir se entran desde teclado y se insertan por orden alfabético ascendente en un "árbol binario". Cada nudo del árbol tiene dos punteros que lo unen a elementos "menores" o "mayores". El recorrido del árbol se realiza comparando los datos nuevos con el campo Nombre de cada nudo y tomando el campo de enlace adecuado (RamifInf o RamifSup). Cuando hallamos un nudo vacío (es decir, uno que tenga una rama de valor NIL), se inserta el elemento. La escritura de los datos en el archivo se consigue entonces de forma sencilla y natural mediante un procedimiento recursivo. Usted podría utilizar estos ejemplos como base para un potente administrador de datos: un juego de programas, quizá, diseñado a medida para su propio uso. Una advertencia: si lee un archivo ya clasificado sobre un árbol binario, cada inserción se producirá en la misma rama y el árbol se convertirá en una lista de enlace simple. Para aplicaciones más simples, probablemente lo más versátil sea la lista circular. En ella no hay punteros NIL en absoluto, ahorrándose, por tanto, una comprobación doble en cada comparación, mientras que el árbol binario tendrá (Cantidad de Nudos+1) NILs.

```
PROGRAM   ArbolClasificador (input, output, ArchivoDatos);

CONST
          NombreArchivo      = 'Arboldatos';
          LongitudSerie      = 25;

TYPE
  Cardinal            = 0..MaxInt;
  TamañoSerie         = 1..LongitudSerie;
  serie               = PACKED ARRAY [TamañoSerie] OF char;
  cosa                = RECORD
                          Nombre    :serie:
                          {otros campos..}
                          deuda     :Cardinal
                        END;  {cosa}
  TipoArchivo         = FILE OF cosa;
  arbol               = ↑ rama;
                        {**referencia adelantada a:}
  rama                = RECORD
                          item            ;cosa;
                          RamifInf,
                          RamifSup        :arbol
                        END;    {rama}

VAR
  datos               : cosa;
  ArchivoDatos        : TipoArchivo;
  tronco              : arbol;

INCLUDE 'Utils.src' {**archivo fuente que contiene :SaltarBlancos,
                    LeerFicha y LeerLinea - ver programa ListaCirc**}
        {1111111111111111111111111111111111111111}

PROCEDURE   Leer Cantidad (VAR cantidad : Cardinal);
        {**Validar un valor=Cardinal legal**}
  VAR
        OK          : boolean;
  BEGIN
    REPEAT
      REPEAT
        write ( 'Cantidad ? ': 20 );

        IF EoLn ( input ) THEN
          ReadLn ( input );


        SaltarBlancos ( input )
        {** volver a solicitarla si final linea **}
      UNTIL NOT EoLn ( input );

      LeerFicha ( input, cantidad, OK );


      IF NOT OK THEN
        WriteLn     ( '---ERROR---' : 20,
                    'por favor vuelva a entrar' )
    UNTIL OK        {**insistir en un numero valido**}

  END;    {LeerCantidad}
          {1111111111111111111111111111111111111111}

PROCEDURE     Crecer      (VAR     hoja     : arbol;
    {**añadir al arbol**}          datos    :cosa);
  BEGIN
    new ( hoja );

    WITH hoja ↑ DO
      BEGIN
        item := datos;
```



Este algoritmo puede dar lugar a errores o a confusión. ¿Para qué necesitamos la variable booleana local hallada? Y ¿por qué debemos asignarle a Longitud el valor N−1? Éstas son molestias mínimas; pero si recurrimos a dispositivos artificiales como éste, la formulación de algoritmos más complejos puede volverse muy confusa y sujeta a errores.

La utilización de "banderas de bits" (que conocemos como tipos booleanos) es el ardid más antiguo del libro del programador, pero a menudo se introducen simplemente por el bien del ordenador y no como parte natural del algoritmo. Debe utilizarse la variable local N, no Longitud, dado que ésta es un identificador de función y no una variable. Los identificadores de función en el lado derecho de una sentencia como:

```
Longitud:=Longitud+1
```

intentarían realizar una llamada recursiva a la propia función Longitud.

Compare esto con la función Longitud necesaria para series que utilicen una lista enlazada. Con la representación dinámica, recuerde que la definición TYPE de serie es muy diferente, permitiendo la creación de una longitud de serie cualquiera. Con frecuencia, las descripciones de datos como éstas se definirán de forma recursiva.

Recursión

Muchos de los ejemplos triviales que se emplean para ilustrar la recursión se podrían expresar igualmente bien (si no mejor) como algoritmos iterativos; pero veamos la función Longitud para el tipo de serie recursiva:

```
FUNCTION Longitud (S:serie)        :Cardinal;
  BEGIN
    IF S=NIL
      THEN
         Longitud:=0
      ELSE
         Longitud:=succ(longitud(S ↑ .siguiente))
  END; {Longitud}
```

La cabeza de la lista es S y la expresión S ↑ .siguiente selecciona el campo puntero del siguiente registro de la lista. Por otra parte, se puede pensar que se trata de una lista que comienza por el siguiente registro (ButFirst). Siempre que no encontremos un NIL, llamamos a una evaluación de la longitud de ButFirst y la incrementamos con succ.

Existen muchos problemas más que sólo se pueden resolver de forma eficaz mediante el empleo natural de la recursión. El ejemplo anterior es más bien trivial y le permitirá escribir Longitud con rela-

```
      RamifInf := NIL; {**No mas datos aun,}
      RamifSup := NIL {mas alla de esta rama**}
   END

   END;    {Crecer}
           {1111111111111111111111111111111111111111}

PROCEDURE          Trepar               ( brote          : arbol;
  {**hallar lugar para insertar**} datos                 : cosa );
  VAR
          tallo              : arbol;
          masgrande          : boolean;
  BEGIN
     WHILE brote<>NIL DO
        BEGIN                {**hallar una rama vacia**}
          tallo := brote;
          masgrande:=datos . Nombre>
                         tallo ↑ .item . Nombre;
          IF masgrande
            THEN             {**trepar hacia arriba**}
              brote := brote ↑ .RamifSup
            ELSE             {**trepar hacia abajo**}
              brote := brote ↑ . RamifInf
        END;

     Crecer ( brote, datos );

     IF masgrande
       THEN                  {**insertar arriba**}
         tallo ↑ .RamifSup := brote
       ELSE                  {**insertar abajo**}
         tallo ↑ .RamifInf := brote
  END;    {Trepar}
          {1111111111111111111111111111111111111111}

PROCEDURE          Imprimir ( VAR      F          : TipoArchivo;
  {**escribir datos en un archivo**} raiz         : arbol );
  BEGIN
     IF raiz<>NIL THEN
        WITH raiz ↑ DO       {**recursion:}
          BEGIN   {**escribir por orden :}
            Imprimir      ( F, RamifInf ); {**primero abajo,}
            write         ( F, item ); {la raiz misma,}
            Imprimir      ( F,RamifSup ) {por ultimo abajo**}
          END

  END;    {Imprimir}
          {1111111111111111111111111111111111111111}

PROCEDURE          Reclamar ( VAR      raiz       : arbol );
  {**recuperar memoria asignada para uso ulterior**}
  BEGIN
     IF raiz<>NIL THEN {**recursion**}
       BEGIN             {**reclamar primero ramificaciones-}
         Reclamar        ( raiz ↑ .RamifInf );
         Reclamar        ( raiz ↑ .RamifSup );
         dispose         ( raiz ) {-despues la raiz**}
       END

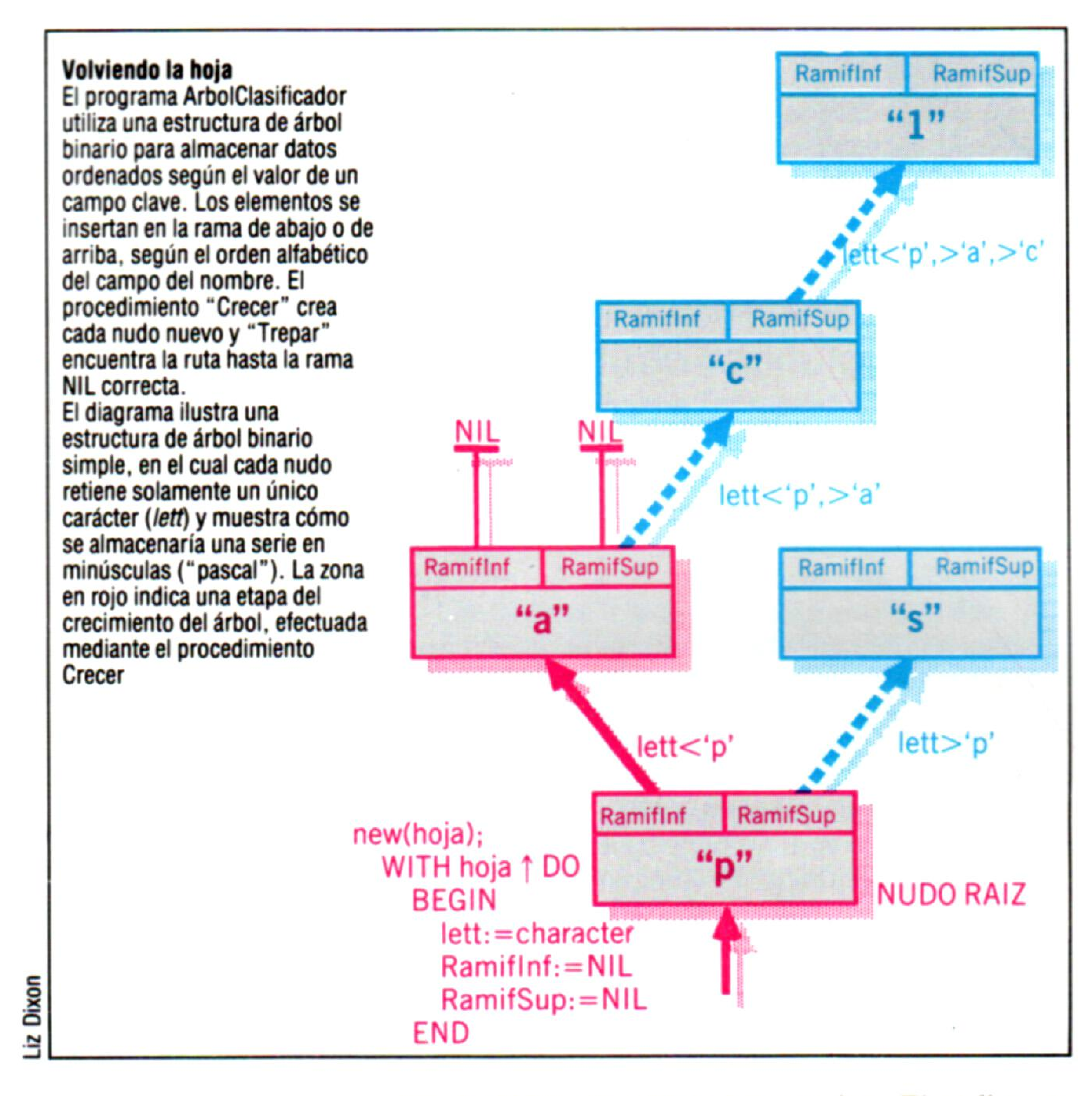
  END;    {Reclamar}
          {1111111111111111111111111111111111111111}

BEGIN         {ArbolClasificador - Programa principal }

  asignar ( ArchivoDatos, NombreArchivo );
  pagina ( output );
  WriteLn ( '=== Clasificacion en arbol ===' : 25 );
  WriteLn;
  WriteLn ( 'Entre algunos datos ( apellido primero)' );
  write ( 'Nombre ? ' );
  LeerLinea ( datos . Nombre );
  LeerCantidad ( datos . deuda );
  Crecer ( tronco, datos ); {*plantar el tronco del arbol*}
  write ( 'Nombre ? ' );
  LeerLinea ( datos. Nombre );

  WHILE datos . Nombre [ 1 ]<>chr ( 0 ) DO
    BEGIN
        LeerCantidad ( datos . deuda );
        Trepar ( tronco, datos );
        WriteLn ( 'RETURN cuando hecho' : 40 );
        write ( 'Nombre ? ' );
        LeerLinea ( datos . Nombre)
    END;
    {**otro proceso, luego escribir el archivo :}
  WriteLn ( 'Escribir los datos en : ',Nombre Archivo );
  rewrite ( ArchivoDatos );
  Imprimir ( ArchivoDatos, tronco );
  WriteLn;
  WriteLn ( 'Deuda' : 8, 'Nombre' : LongitudSerie );
  WriteLn;
  reset      ( ArchivoDatos ); {**leer el archivo,}
             {e imprimir los datos ordenados :}
  WHILE NOT EoF ( ArchivoDatos ) DO
    BEGIN                {**leer cada registro**}
        read ( ArchivoDatos, datos );
          {**escribir los campos en la VDU : }
        WITH datos DO
          WriteLn ( deuda : 8, ' : ', Nombre )
    END;

  Reclamar ( tronco ); {**disponer de la memoria}
     {para posteriores procesos, etc.**}
END.
```



Volviendo la hoja
El programa ArbolClasificador utiliza una estructura de árbol binario para almacenar datos ordenados según el valor de un campo clave. Los elementos se insertan en la rama de abajo o de arriba, según el orden alfabético del campo del nombre. El procedimiento "Crecer" crea cada nudo nuevo y "Trepar" encuentra la ruta hasta la rama NIL correcta.
El diagrama ilustra una estructura de árbol binario simple, en el cual cada nudo retiene solamente un único carácter (*lett*) y muestra cómo se almacenaría una serie en minúsculas ("pascal"). La zona en rojo indica una etapa del crecimiento del árbol, efectuada mediante el procedimiento Crecer

tiva facilidad sin utilizar la recursión. El código es más extenso y se debe emplear un contador local (al igual que en la versión de matriz) específicamente para evitar la recursión.

```
FUNCTION Longitud (S:serie)   :Cardinal;
  VAR
    N   :Cardinal;
  BEGIN
    N:=0;
    WHILE S<>NIL DO
      BEGIN
        N:=N+1;
        S:=S ↑ .siguiente
      END;
    Longitud:=N
  END;   {Longitud}
```

Incluso en este algoritmo iterativo la claridad y la precisión son evidentes y obedecen a la simple naturaleza recursiva de la estructura de datos.

Muchos compiladores de PASCAL soportan *directivas*, que son instrucciones para el compilador, *no* declaraciones ni sentencias. En realidad la única directiva exigida por el ISO Standard es Forward (más adelante). En el caso de que dos procedimientos o funciones necesiten llamarse entre sí, se dice que son *mutuamente recursivos*. Esto se produce muy raramente, pero plantea un problema: no se puede utilizar ningún objeto del PASCAL hasta que haya sido declarado o definido.

La solución estriba en declarar sólo el encabezamiento de un subprograma, sustituyendo su bloque por la directiva del compilador FORWARD. Tras la definición completa del otro módulo, se consigna el encabezamiento de forma abreviada (omitiendo la lista de parámetros) y es entonces cuando se define el bloque. A menudo se suele disponer de otras directivas para controlar las opciones de compilación, pero no deben emplearse de forma liberal si se desea conservar la portabilidad. Esto significa que, además de que como primer carácter del comentario debe aparecer un símbolo especial (por lo general $), las opciones no portables podrán no ser aceptadas por un compilador diferente.

Más complementos

Algunas implementaciones no estandarizadas (en particular HiSoft) exigen una sintaxis ligeramente distinta para adelantar declaraciones de puntero. En este caso, usted deberá remitirse a su manual. Hay muy pocos "complementos" de esta clase, y ninguno en absoluto para los compiladores de PASCAL que se ajusten a la definición ISO del lenguaje.

Otra diferencia que quizá encuentre en las versiones UCSD, TCL/RML/Oxford y HiSoft es la falta de un procedimiento dispose. En lugar del mismo, se proporcionan los procedimientos no estandarizados mark y release.

Existe aún otra importante descripción avanzada de datos en PASCAL que aún no hemos mencionado: la "variante". Cuando hemos querido almacenar elementos de tipos diferentes, hemos utilizado un registro con campos de tipos adecuados. Pero supongamos que es necesario que la descripción de una parte, o incluso de todo el registro, sea flexible. Típicamente, puede que se desee almacenar información personal diferente sobre los llamados "sujetos de datos", que dependa, por ejemplo, de si el individuo está casado o no. En un registro variante se define primero su "parte fija", después se especifica la parte variante mediante la introducción de un selector de variante (de cualquier tipo simple) y utilizando las palabras reservadas CASE y OR:

```
TYPE
  genero=(varon,mujer);
  variante=RECORD
    {cualquier campo comun}
    CASE casado:boolean OF
      false:();
      true :(FechaCasamiento:serie;
        CASE sexo:genero OF
          varon:();
          mujer:(NombreSoltera:serie))
        END:{variante}
```

Observe que las listas de campos vacíos deben tener paréntesis. El espacio en un archivo será fijo (para la variante más larga), pero se puede guardar memoria con punteros de variantes tales como new(p,true,varon).

Los escasos puntos débiles del PASCAL en gran parte han sido eliminados mediante los esfuerzos de los especialistas en PASCAL (y por Wirth en el MODULA-2). El lenguaje es formidablemente potente y, aun así, pequeño, eficaz, para fines generales y fácil de aprender. Por supuesto, si usted se ciñe a la definición ISO habrá algunas operaciones a nivel de sistema que no podrá implementar. Una posible solución es escribir estas rutinas en ensamblador o BCPL y unirlas a un programa en PASCAL. Las ventajas de que todo su código fuente en PASCAL sea portable a cualquier micro, mini u ordenador central del mundo son indiscutibles. El PASCAL es el enfoque más próximo que tenemos a una *lingua franca* informática.



Un compatible competitivo



Chris Stevens

Alcanzando al gigante
El Tandy 1000 es una máquina compatible con el IBM-PC, con el que la empresa confía recuperar parte del mercado que le ha sido arrebatado por la intervención de la gigantesca IBM Corporation. Para conseguir la compatibilidad, el Tandy 1000 está basado en el procesador Intel 8088 y utiliza las unidades de disco estándar de 5 ¼ pulgadas. En la fotografía vemos el modelo de unidades de disco gemelas, que posee 128 K adicionales de memoria. El ordenador está visualizando el *Flight simulator* (simulador de vuelo) de Microsoft, escrito para el IBM-PC. La capacidad para ejecutar este programa es una prueba de su compatibilidad

Con la introducción del modelo 1000 la empresa Tandy espera ofrecer una alternativa relativamente económica al IBM-PC

Aunque Tandy Corporation fue uno de los primeros fabricantes que se introdujo en el mercado del microordenador, con el TRS-80, la empresa no consiguió ganar el interés masivo que obtuvo el Commodore en el mercado personal o el Apple en el mercado de gestión. Ahora Tandy parece haber modificado sus planteamientos comerciales y se dispone a un ataque bilateral sobre el mercado de gestión. Por una parte, asociándose con ACT (fabricantes del Apricot), Tandy ha puesto al día su gama de máquinas en la cadena de centros de informática de toda Europa, que están ahora vendiendo la gama Apricot. La otra punta de lanza de la ofensiva de la empresa es la que representa su propio departamento de fabricación.

A muchas personas de la industria durante cierto tiempo les ha parecido obvio que quienquiera que pudiera ofrecer una máquina verdaderamente compatible con el IBM-PC a un precio reducido, estaría llamado a ser un ganador, tanto en el mercado de gestión como en el mercado del ordenador personal de Estados Unidos, donde los consumidores tienden a adquirir máquinas para uso personal que en Europa se consideran demasiado caras para cualquier aplicación que no sea de gestión.

Se afirma que el Tandy 1000 es una máquina totalmente compatible y, con una única unidad de disco y 128 K cuesta poco más de £1 100 (unas 240 000 ptas) en el mercado británico. Sin duda alguna, Tandy confía en que, al rebajar drásticamente su precio frente al de la competencia, podrá revitalizar su departamento de productos de informática.

El ordenador que vamos a examinar aquí es la versión del Tandy 1000 de 256 K con unidades de disco gemelas. A primera vista, guarda un parecido más que casual con el IBM-PC. La máquina se compone de una caja grande que contiene el ordenador propiamente dicho, las interfaces y las unidades de disco. Encima se apoya la pantalla y hay un teclado móvil que se puede colocar del modo más conveniente.

La máquina Tandy posee el mismo aspecto sólido y fiable que el IBM-PC. Al igual que la máquina IBM, el Tandy 1000 incorpora un teclado que se enchufa al ordenador y que tiene teclas esculpidas y una superficie curvada para facilitar la digitación. Además, para contribuir a un posicionamiento óptimo, hay debajo dos patas que inclinan el teclado hasta un ángulo de 15°.

Si bien el Tandy posee las teclas necesarias para lograr la compatibilidad, éstas están dispuestas de un modo distinto al del teclado del IBM-PC. Esta estrategia tiene sus ventajas y sus inconvenientes. En el lado positivo, el teclado del IBM, aunque alabado por su diseño, ha sido categóricamente criticado en muchos sectores por una cierta inconveniencia en la colocación de algunas teclas vitales, como las teclas Shift y Alternate. El tamaño relativamente pequeño de las teclas Return y Control también ha suscitado críticas.

Es obvio que Tandy ha tomado nota de estas críticas en el momento de diseñar el teclado. La tecla "/", que causaba muchísimos problemas al utilizar la tecla Shift del lado izquierdo, ahora se ha suprimido por completo, asumiendo sus funciones las teclas de SHIFT 4 y 7 del teclado numérico. Del mismo modo, las teclas Alternate y Caps Lock se han trasladado desde sus posiciones a los lados de la barra

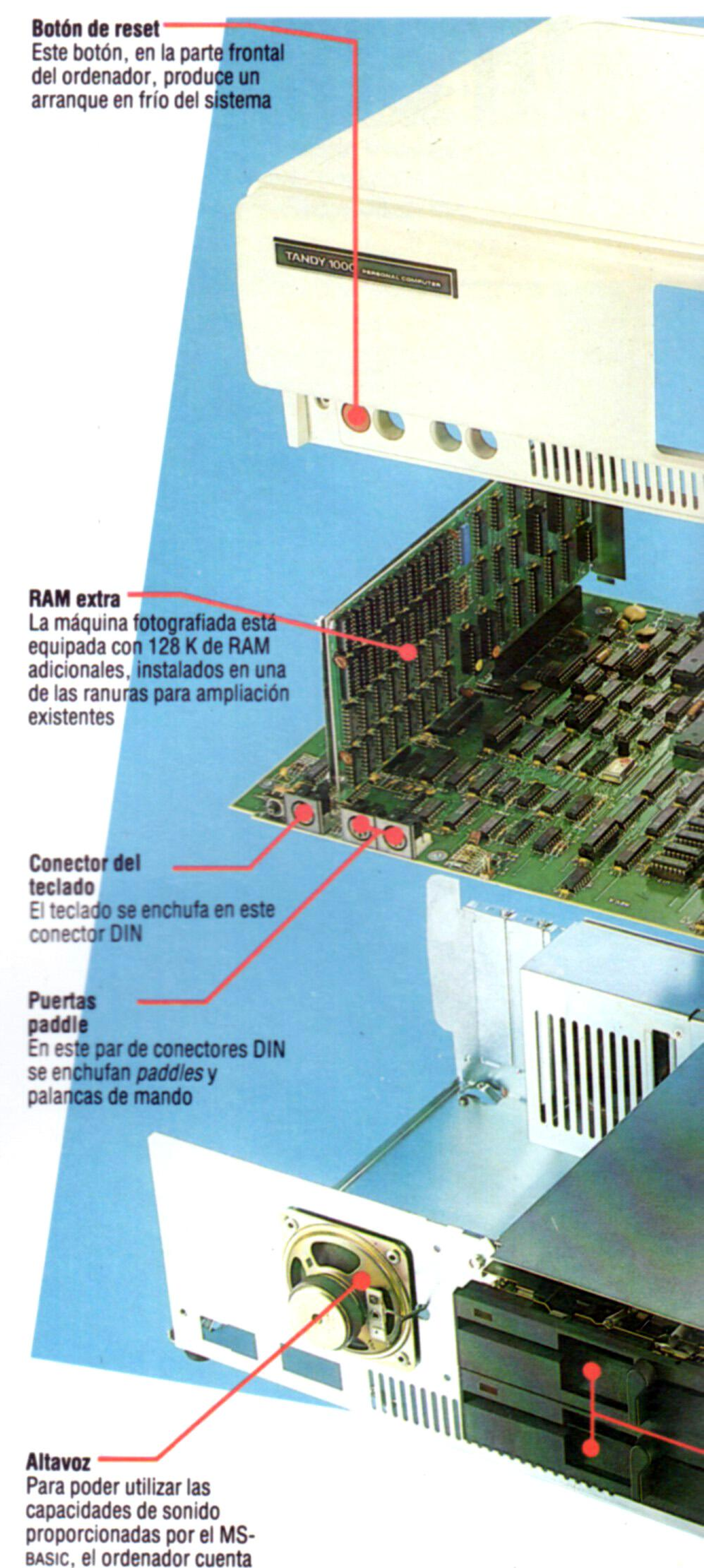
espaciadora hasta la zona del teclado numérico y abajo y a la izquierda de las teclas Control, respectivamente. Estos cambios han tenido como efecto global que la digitación normal resulte mucho más sencilla que en el teclado IBM.

La desventaja de estos cambios formales es, por supuesto, que una vez que uno se ha tomado la molestia de familiarizarse con el idiosincrásico trazado del IBM-PC, tendrá que volver a aprenderse el teclado. No obstante, a los recién iniciados les resultará mucho más fácil habituarse al mismo. Las teclas son de recorrido total y su calidad "táctil" es una de las más estimables del mercado.

Asimismo, Tandy ha introducido otros cambios en el teclado. Las teclas Insert y Delete se han incorporado con las funciones + y − y ahora se hallan encima del teclado, no debajo. Entre las adicionales se incluyen teclas individuales del cursor para edición, así como teclas Hold, Print y Break, todas las cuales se han situado entre las de máquina de escribir y las de teclado de calculadora. La adición de estas teclas ha supuesto la reubicación de las teclas Function, desde el extremo izquierdo del teclado en el IBM-PC, hasta justo encima de las de máquina de escribir. Aunque probablemente haya sido inevitable, ahora las teclas están situadas de forma menos conveniente que lo que lo estaban antes, pero, a modo de compensación, Tandy ha proporcionado dos teclas Function adicionales.

Retornando al ordenador propiamente dicho, Tandy ha optado por la forma de caja grande propiciada por IBM. En la parte frontal, en el lado izquierdo, hay un par de unidades de disco flexible de 5 ¼ pulgadas, si bien el modelo estándar posee una sola unidad pero con facilidad para albergar otra. Aunque en líneas generales son algo más silenciosas que sus equivalentes IBM, las unidades mostraron pequeñas diferencias en cuanto a velocidad de acceso. Una molestia mínima es que los discos no se cargan a resorte. Esto significa que no saltan fuera de las unidades cuando éstas se destraban y que hay que sacarlos con los dedos.

Debajo de la proyección frontal, junto a una rejilla de ventilación, hay un par de enchufes DIN de 270° y seis patillas para palancas de mando u otros dispositivos de control externo. A la izquierda de éstos hay un botón grande de color naranja, la reinicialización del sistema, que proporciona el arranque en frío. A pesar del hecho de estar tan a la vista, existen muy pocas probabilidades de pulsarlo



Botón de reset
Este botón, en la parte frontal del ordenador, produce un arranque en frío del sistema

RAM extra
La máquina fotografiada está equipada con 128 K de RAM adicionales, instalados en una de las ranuras para ampliación existentes

Conector del teclado
El teclado se enchufa en este conector DIN

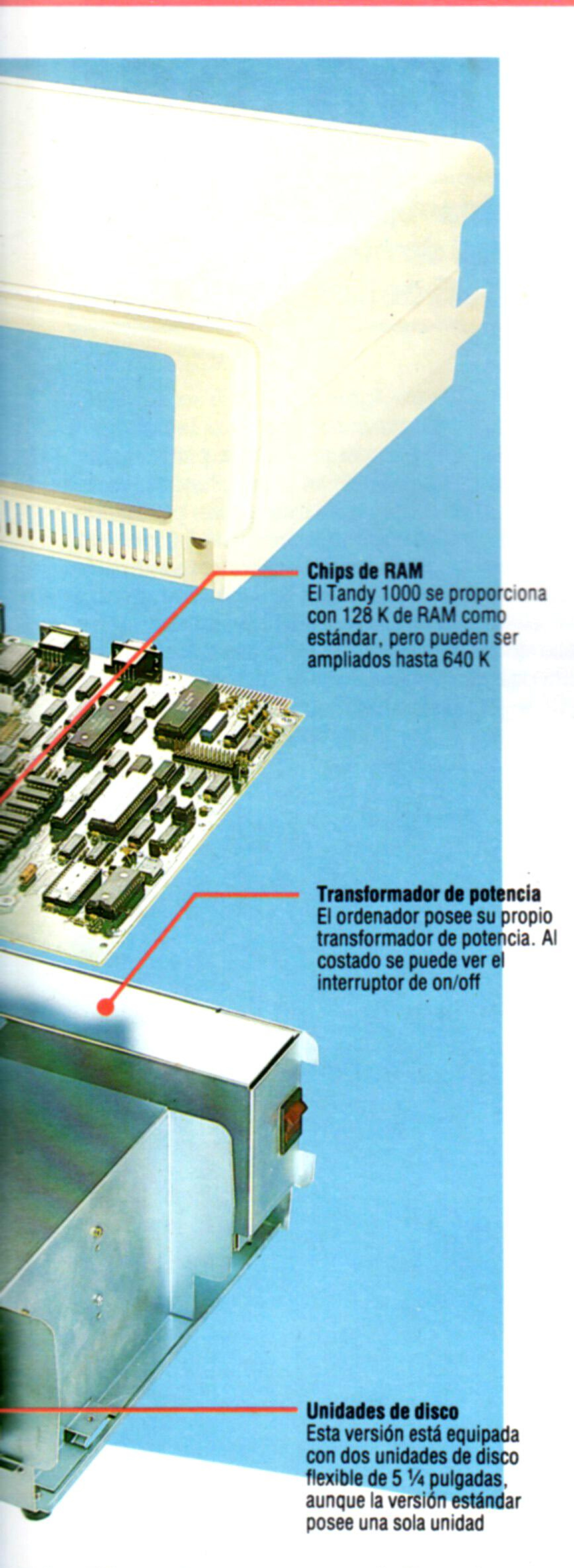
Puertas paddle
En este par de conectores DIN se enchufan *paddles* y palancas de mando

Altavoz
Para poder utilizar las capacidades de sonido proporcionadas por el MS-BASIC, el ordenador cuenta con un altavoz incorporado

La parte trasera
Además de contar con una interface para impresora Centronics, el Tandy 1000 posee los dos tipos de conector para pantalla: RGB y video compuesto. Con el fin de aprovechar al máximo las capacidades de sonido del Tandy hay, asimismo, un enchufe de audio que se proporciona para amplificar el sonido a través de un sistema de alta fidelidad convencional

Chris Stevens





Chips de RAM
El Tandy 1000 se proporciona con 128 K de RAM como estándar, pero pueden ser ampliados hasta 640 K

Transformador de potencia
El ordenador posee su propio transformador de potencia. Al costado se puede ver el interruptor de on/off

Unidades de disco
Esta versión está equipada con dos unidades de disco flexible de 5 ¼ pulgadas, aunque la versión estándar posee una sola unidad

inadvertidamente, puesto que se halla en la carcasa. De hecho, quizá sea la mejor posición posible, porque numerosas aplicaciones no permiten que usted salga al sistema operativo sin pulsar antes este botón. Por consiguiente, resulta útil un *reset* situado convenientemente, en especial en una máquina tan grande.

En la parte trasera del ordenador están las interfaces para periféricos. En el extremo izquierdo se halla la entrada de la fuente de alimentación eléctrica, justo encima de la interface en paralelo Centronics, proporcionada como conector marginal. A la derecha hay un conector D de siete patillas para un lápiz óptico y una puerta para la conexión de un monitor RGB. Más hacia la derecha, debajo del ventilador de refrigeración, hay un par de conectores microjack, uno de los cuales proporciona un conector de video compuesto para pantallas monocromáticas o a color de video compuesto; el otro es un conector de audio que puede amplificar el sonido del ordenador a través de un sistema de alta fidelidad convencional. En el extremo derecho hay tres conectores de ampliación en los que se pueden instalar interfaces para periféricos adicionales o placas de memoria.

Méritos comparativos

Tandy ha hecho hincapié en el hecho de que su máquina proporciona muchas facilidades de interface de las que no dispone el IBM-PC estándar. Aunque esto es verdad, la empresa parece haber proporcionado apenas el mínimo necesario para una máquina de gestión utilizable. Por ejemplo, la máquina de precio mínimo está dotada de 128 K de RAM, lo que no es suficiente para ejecutar algunos de los paquetes de software integrado más nuevos. Y, si bien el Tandy 1000 es una máquina completamente autosuficiente, una puerta RS232 (que es opcional) es esencial para cualquier clase de comunicaciones que pueda requerir el usuario de gestión. Por otra parte, ciertamente ofrece, considerando su precio, más prestaciones que el IBM-PC, el cual, con las mismas facilidades, costaría mucho más. Siempre y cuando, por supuesto, el Tandy sea auténticamente compatible con el software IBM.

El problema de intentar la compatibilidad con el IBM-PC no estriba ni en la CPU 8088 y ni siquiera en el sistema operativo MS-DOS, dado que ambos se pueden adquirir contactando con los fabricantes adecuados. La dificultad estriba en el BIOS (Basic Input/Output System: sistema básico de entrada/salida), cuyo *copyright* pertenece a IBM. Muchos programas emplean saltos directos a las rutinas BIOS y, a menos que una máquina compatible tenga las rutinas relevantes en las mismas direcciones, el programa no funcionará correctamente.

En este sentido, el BIOS del Tandy 1000, escrito por Phoenix Compatibility Corporation, es excepcional. En la máquina no sólo se ejecuta el *Lotus 1-2-3*, cuyas dificultades son notorias (aunque aún con 256 K de espacio de memoria estaba bastante justo), sino que el ordenador también ejecuta el *Wordstar 2000* y el *dBase II*.

A pesar de ello, el Tandy 1000, al igual que el IBM-PC, es sumamente lento para cálculos matemáticos en BASIC. Completar un contador de 0 a 1000, utilizado como sencillo programa comparativo, llevó seis segundos completos, más que muchos micros de ocho bits.

El Tandy 1000 es, indudablemente, una alternativa viable para el usuario de gestión que desee sacar partido de la gran cantidad de software de calidad que se ha escrito para el IBM-PC. Sin embargo, debido a las limitaciones de hardware que impone la compatibilidad con IBM y a la necesidad de mantener el precio lo más bajo posible, Tandy ha presentado una máquina que, desde el punto de vista de la informática de gestión, es esencialmente anticuada. Así y todo, siempre y cuando IBM no reduzca el precio de su máquina a modo de represalia (lo que parece improbable), la Tandy Corporation habrá producido una máquina que obtendrá un enorme éxito.

TANDY 1000

DIMENSIONES
420×335×150 mm

CPU
8088, operando a 4,77 MHz

MEMORIA
128 K de RAM como estándar; ampliables a 640 K

PANTALLA
80×25 caracteres en modalidad de texto, 640×200 pixels en alta resolución. La máquina puede visualizar hasta ocho colores entre una gama de 16

INTERFACES
Conectores para monitor compuesto y RGB, puerta para impresora, enchufe de audio y tres interfaces para ampliación

LENGUAJES DISPONIBLES
BASIC, más una selección disponible bajo MS-DOS incluyendo COBOL, FORTRAN, etc.

TECLADO
90 teclas en total, incluyendo un teclado numérico y 12 teclas de función

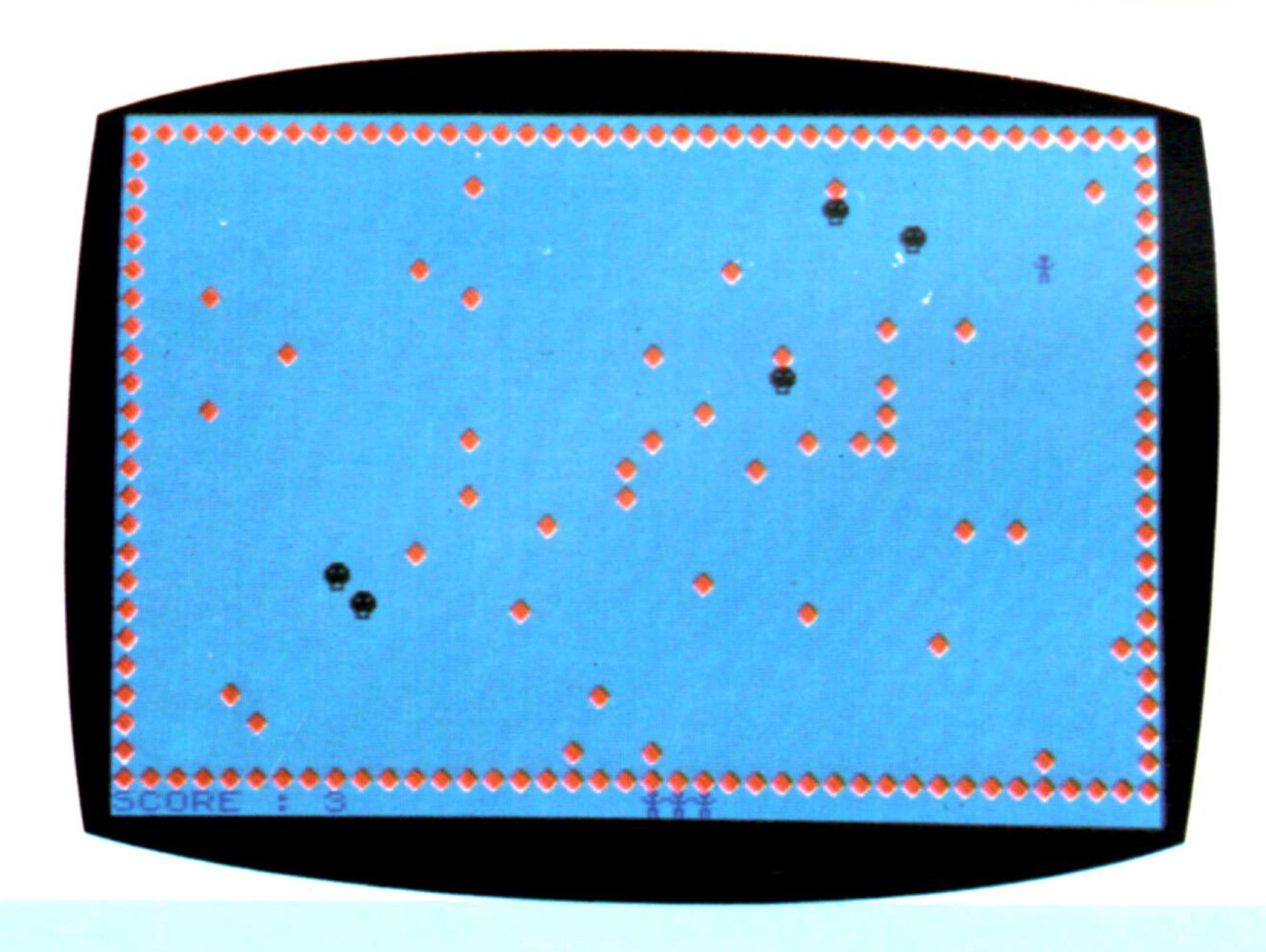
DOCUMENTACION
La documentación incluye explicaciones del MS-DOS, MS-BASIC y el paquete integrado Deskmate que se entrega junto con la máquina

VENTAJAS
Totalmente compatible con el IBM-PC, proporcionando una cantidad de interfaces de las que no dispone el modelo IBM básico

DESVENTAJAS
Con el diseño de la máquina, Tandy no ha avanzado más allá del diseño del IBM-PC original. La utilización de un procesador 8088 en lugar del 8086, más avanzado, significa que la operación del Tandy 1000 es más lenta de lo que cabía esperar

Robots en el MO5

Usted se encuentra solo, abandonado sobre un planeta defendido por robots asesinos. El suelo está sembrado de minas... Ya conoce el problema. Pero tal vez no en el MO5 de Thomson



Las minas están representadas en la pantalla por rombos rojos. Al comenzar el juego, hay cinco robots preparados sobre el terreno. Sin perder un solo segundo, se precipitan sobre usted siguiendo siempre el camino más corto. Afortunadamente, son ciegos y no pueden ver las minas situadas entre usted y ellos, lo cual le permitirá, siempre que se desplace de forma adecuada, eliminarlos. Para ello utilice la palanca de mando o las teclas A, Z, E, Q, D, W, X, C, según la dirección elegida por usted. Una vez eliminados todos los robots, el juego prosigue con un robot suplementario. Si salta sobre una mina o un robot le mata, aún no está todo perdido. Usted dispone de cinco vidas. Si desea cambiar el número de minas, modifique el valor de la variable NM en la línea 80.

```
10 REM *************
20 REM *   ROBOTS   *
30 REM *************
40 DEFINT A-Z
50 CLEAR ,,3
60 NH=5
70 N1=5
80 NM=40
90 NR=N1
100 DIM R(30,1)
110 GOSUB 1580
120 GOSUB 1470
130 GOSUB 910
140 DN JS GOSUB 710,810
150 C=POINT(HX*8+4,HY*8+4)
160 IF C<>-13 AND C<>4 THEN 470
170 COLOR 4
180 LOCATE X,Y
190 PRINT N$;
200 LOCATE HX,HY
210 PRINT H$;
220 X=HX
230 Y=HY
240 T=0
250 FOR I=1 TO NR
260 IF R(I,0)=0 THEN 400
270 T=1
280 RX=R(I,0)+SGN(HX-R(I,0))
290 RY=R(I,1)+SGN(HY-R(I,1))
300 C=POINT(RX*8+4,RY*8+4)
310 IF C=1 OR C=0 THEN S=S+1:LOCATE R(I,
    0),R(I,1):PRINT N$;:R(I,0)=0:GOTO 400
320 IF C=4 THEN 470
330 COLOR 0
340 LOCATE R(I,0),R(I,1)
350 PRINT N$;
360 LOCATE RX,RY
370 PRINT R$;
380 R(I,0)=RX
390 R(I,1)=RY
400 NEXT I
410 IF T=0 THEN 430
420 GOTO 140
430 S=S+10
440 IF INKEY$<>"" THEN 440
450 IF NR<30 THEN NR=NR+1
460 GOTO 130
470 NH=NH-1
480 COLOR 7
490 LOCATE X,Y
500 PRINT N$;
510 LOCATE HX,HY
520 PRINT H$;
530 PLAY "L96REL72REL24REL96REL72FAL24MI
    L72MIL24REL72REL24DO#L96RE"
540 IF INKEY$<>"" THEN 540
550 IF NH>0 THEN NR=N1:GOTO 130
560 CLS
570 SCREEN 1,6,6
580 ATTRB 1,1
590 LOCATE 9,10
600 PRINT "PUNTOS :";S;
610 LOCATE 9,20
620 PRINT "OTRA ?";
630 COLOR 4
640 ATTRB 0,0
650 IF INKEY$<>"" THEN 650
660 D$=INKEY$
670 IF D$="" THEN 660
680 IF D$<>"N" THEN RUN
690 CLS
700 END
710 D$=INKEY$
720 IF D$="A" THEN HX=HX-1:HY=HY-1
730 IF D$="Z" THEN HY=HY-1
740 IF D$="E" THEN HY=HY-1:HX=HX+1
750 IF D$="Q" THEN HX=HX-1
760 IF D$="D" THEN HX=HX+1
770 IF D$="W" THEN HX=HX-1:HY=HY+1
780 IF D$="X" THEN HY=HY+1
790 IF D$="C" THEN HY=HY+1:HX=HX+1
800 RETURN
810 J=STICK(0)
820 IF J=1 THEN HY=HY-1
830 IF J=2 THEN HY=HY-1:HX=HX+1
840 IF J=3 THEN HX=HX+1
850 IF J=4 THEN HX=HX+1:HY=HY+1
860 IF J=5 THEN HY=HY+1
870 IF J=6 THEN HY=HY+1:HX=HX-1
880 IF J=7 THEN HX=HX-1
890 IF J=8 THEN HX=HX-1:HY=HY-1
900 RETURN
910 CLS
920 COLOR 4
930 LOCATE 0,24
940 PRINT "PUNTOS :";S;
950 IF NH=1 THEN 1000
960 FOR HX=1 TO NH-1
970 LOCATE 19+HX,24
980 PRINT H$;
990 NEXT HX
1000 COLOR 1
1010 FOR HX=0 TO 39
1020 LOCATE HX,0
1030 PRINT M$;
1040 LOCATE HX,23
1050 PRINT M$;
1060 NEXT HX
1070 FOR HY=1 TO 22
1080 LOCATE 0,HY
1090 PRINT M$;
1100 LOCATE 39,HY
1110 PRINT M$;
1120 NEXT HY
1130 FOR I=1 TO NM
1140 HX=INT (RND*38)+1
1150 HY=INT(RND*22)+1
1160 IF SCREEN(HX,HY)<>32 THEN 1140
1170 LOCATE HX,HY
1180 PRINT M$;
1190 NEXT I
1200 COLOR 0
1210 FOR I=1 TO NR
1220 R(I,0)=INT(RND*38)+1
1230 R(I,1)=INT(RND*22)+1
1240 IF SCREEN(R(I,0),R(I,1))<>32 THEN
     1220
1250 LOCATE R(I,0),R(I,1)
1260 PRINT R$;
1270 NEXT I
1280 HX=INT (RND*38)+1
1290 HY=INT (RND*22)+1
1300 IF SCREEN(HX,HY)<>32 THEN 1280
1310 X=HX
1320 Y=HY
1330 FOR I=1 TO 5
1340 LOCATE HX,HY
1350 COLOR 5
1360 PRINT CHR$(127);
1370 BEEP
1380 FOR J=1 TO 50
1390 NEXT J
1400 LOCATE HX,HY
1410 COLOR 4
1420 PRINT H$;
1430 FOR J=1 TO 50
1440 NEXT J
1450 NEXT I
1460 RETURN
1470 CLS
1480 SCREEN 4,12,0
1490 ATTRB 1,1
1500 LOCATE 10,10,0
1510 PRINT "PAL. MANDO ?";
1520 ATTRB 0,0
1530 D$=INKEY$
1540 C=RND
1550 IF D$="" THEN 1530
1560 IF D$="0" THEN JS=2 ELSE JS=1
1570 RETURN
1580 DEFGR$(0)=28,28,73,62,8,28,20,20
1590 DEFGR$(1)=60,126,219,255,255,126,36 ,60
1600 DEFGR$(2)=0,0,24,60,126,126,60,24
1610 H$=GR$(0)
1620 R$=GR$(1)
1630 M$=GR$(2)
1640 N$=CHR$(32)
1650 RETURN
```

© SYBEX



Nuevo proyecto

Iniciamos la construcción de un trazador digital para utilizar con el BBC Micro. En primer lugar, esbozamos las etapas del proyecto y proporcionamos la lista de componentes

Un trazador digital es, simplemente, un dispositivo mediante el cual se pueden trazar formas y figuras reales sobre un tablero y visualizarlas en una pantalla de ordenador. Una vez en forma digital, una imagen se puede ya sea editar o bien guardar en disco o cinta. El componente básico del dispositivo es un brazo metálico, dividido en dos partes y con unos goznes para conformar una juntura de hombro y codo. Sobre estos goznes hay montados dos potenciómetros que se pueden usar para proporcionar datos sobre los ángulos de cada juntura. Mediante el empleo del convertidor A/D que lleva incorporado el BBC Micro, la salida analógica de cada potenciómetro se puede convertir a forma digital y, tras el calibrado, los datos entrantes se pueden manipular luego matemáticamente para dar la posición del extremo del segundo brazo en relación a la juntura del hombro. Las figuras de las ilustraciones muestran cómo encajan entre sí los elementos básicos de construcción. La unidad se asienta sobre un tablero cuadrado de 46 cm en el cual se puede colocar la figura a trazar. Los brazos de metal se montan sobre una pequeña caja plástica, en la cual va montado, asimismo, un conector DIN

PERILLAS TIPO "VOLUMEN"
PERNO Y TUERCAS PARA MONTAR LA MIRA
POTENCIÓMETROS DE LAS JUNTURAS
CONECTORES DIN A PUERTA ANALÓGICA
TABLERO BASE
INTERRUPTOR LÁPIZ ARRIBA/ LÁPIZ ABAJO
BRAZOS DEL TRAZADOR
MIRA
Kevin Jones

Elementos de trazado
El trazador digital de *Bricolaje* utiliza dos potenciómetros para proporcionar información que se pueda convertir mediante software, merced a la puerta analógica del BBC Micro, para dar la posición de la mira sobre el tablero

de 5 patillas que acepta un cable conector a la puerta analógica del ordenador, así como un interruptor de presión que le proporciona al trazador una opción incorporada de lápiz arriba/lápiz abajo. En la punta del trazador se fija un pequeño trozo de plástico transparente, en cuya superficie hay grabadas señales en cruz para conformar una mira. El sistema de montaje está diseñado de modo tal que la mira se pueda subir o bajar para facilitar el trazado exacto sobre materiales de espesor variable.

Lista de componentes

Cant.	Artículo
2	Potenciómetro lineal de 100 K-ohmios
2	Perilla de 40 mm
1	Interruptor de presión
1	Caja de 114×76×38 mm
1	Enchufe DIN de 5 patillas
1	Conector DIN de 5 patillas
1 m	Cable plano de 10 vías
1	Caja pernos M5
1	Caja pernos M3
1	Caja tuercas M3
1	Caja tornillos sin tuerca
1	Enchufe D de 15 vías
1	Cubierta D de 15 vías

Varios

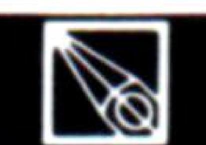
1	Fleje de aluminio de 600×25×2,5 mm*
1	Tablero recubierto de melamina de 46×46 mm**
1	Trozo pequeño de plástico transparente***

Observaciones

*	A adquirir en la mayoría de ferreterías. Se podrían usar también tiras de madera
**	También llamado "Contiplas", se puede asimismo adquirir en ferreterías en diversas anchuras y longitudes
***	Se puede utilizar la sección transparente de un estuche de cassette

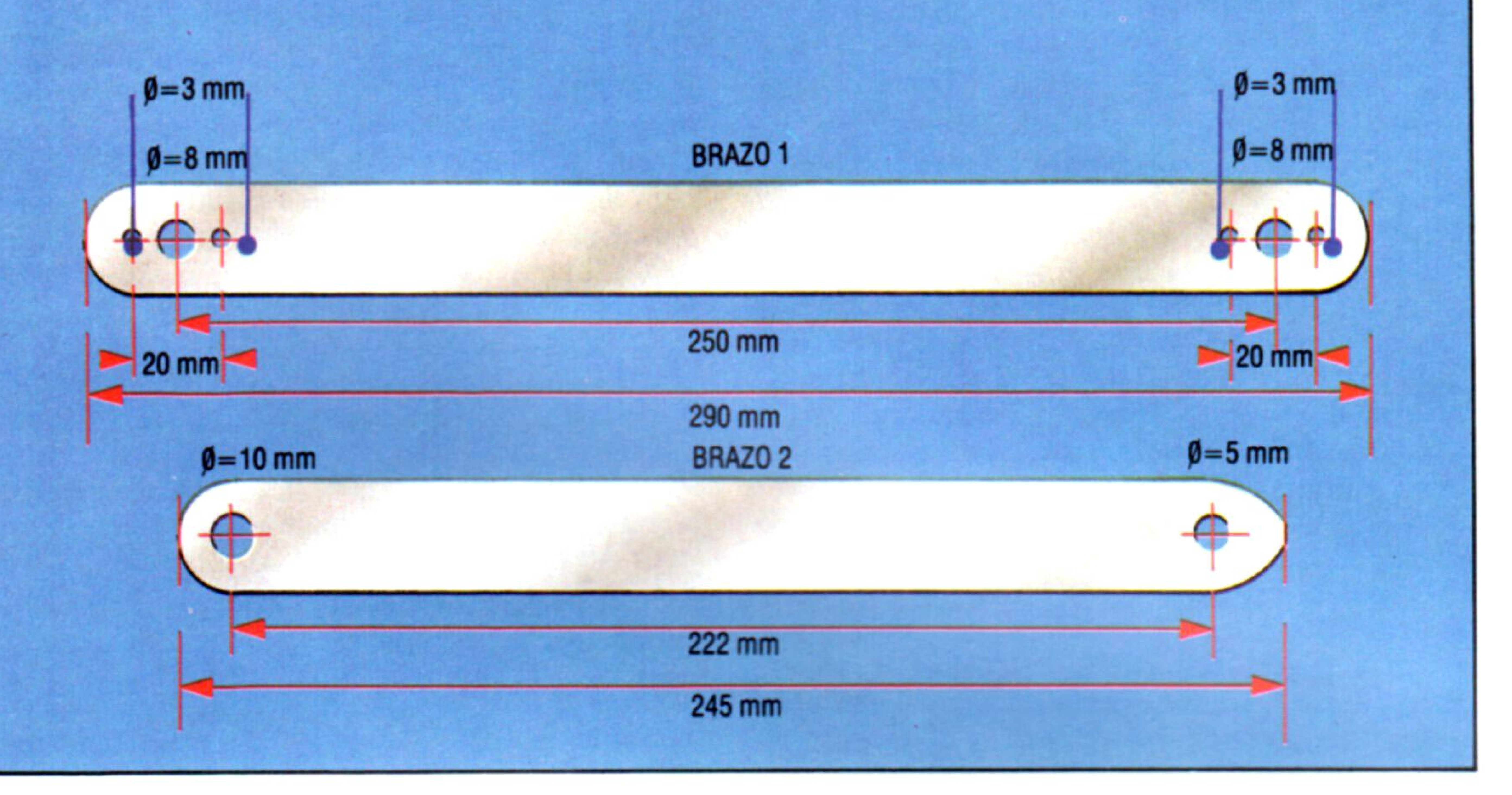
Paso 1: Taladrado de la caja

La primera etapa en la construcción del trazador es, si fuera necesario, cortar el tablero de base al tamaño adecuado. Luego se habrán de perforar en la pequeña caja plástica una serie de agujeros en los que se instalarán, como se aprecia en la ilustración, el brazo del trazador, el interruptor y el conector DIN. Tras haber perforado todos estos agujeros, se puede montar la caja en la esquina superior izquierda del tablero base, dejando un margen libre de 10 mm desde los bordes



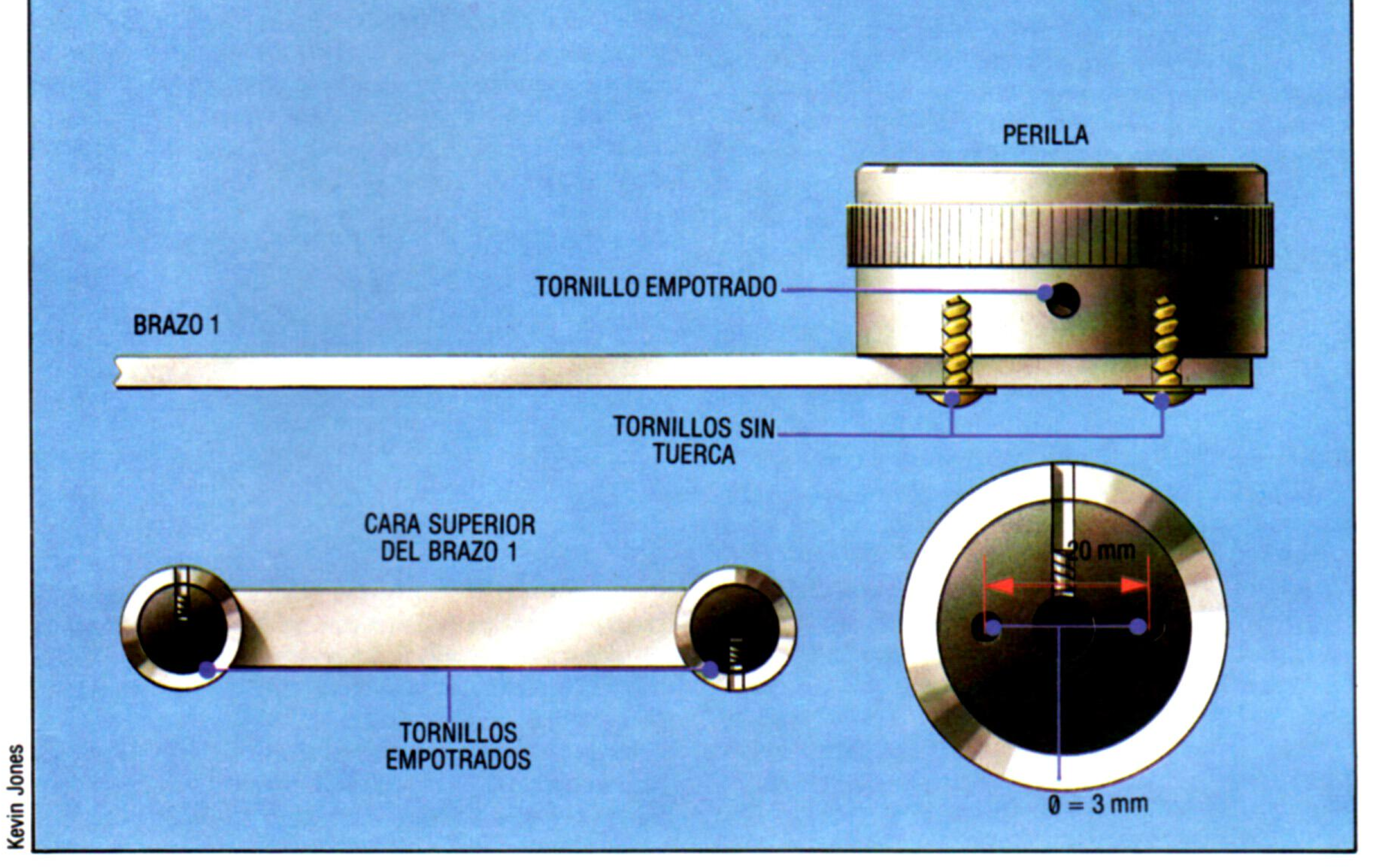
Paso 2: Cortado del brazo

A continuación, corte el fleje metálico formando las dos longitudes de la ilustración y taladre los agujeros requeridos. Observe, sin embargo, que si bien no son imprescindibles las longitudes exactas, sí deben ser precisas las distancias entre los agujeros taladrados en cada extremo de los brazos. Ello se debe a que estas distancias las utilizará el software para llevar a cabo los diversos cálculos geométricos necesarios para determinar la posición de la mira del trazador cuando se use la unidad. Por consiguiente, es importante medir cuidadosamente estas distancias antes de comenzar a perforar



Paso 3: Colocación de las perillas

La última tarea de este capítulo consiste en montar la perilla de "volumen" en cada uno de los extremos del brazo 1. Estas perillas se utilizarán para fijar los potenciómetros que implementaremos en el próximo capítulo. Observe que el husillo del potenciómetro se sujetará mediante un pequeño tornillo empotrado en el costado de la perilla. Se ha de tener cuidado al montar las perillas, de modo que los tornillos empotrados queden orientados en dirección opuesta. Las perillas se fijan en el brazo perforando dos agujeros de 3 mm de diámetro en la base de cada perilla, atravesándolas casi totalmente. Luego se deben instalar las perillas en el brazo 1 con la orientación correcta, utilizando tornillos sin tuerca, como se indica



Kevin Jones

Entramados

Estudiaremos en dos capítulos algunas rutinas de coma flotante del intérprete de BASIC en el Commodore 64

Las rutinas de coma flotante del intérprete de BASIC no están muy bien documentadas (no hay, por ejemplo, una tabla de saltos como la hay para el núcleo), por lo que la búsqueda de las llamadas puede ser una tarea lenta a base de intentos. Para dar a nuestra investigación una meta vamos a tratar de crear una rutina de gráficos en código máquina que nos permita obtener un dibujo en tres dimensiones por medio de líneas y sobre pantalla en alta resolución.

Las ideas matemáticas que aquí exponemos pueden emplearse como base de otras rutinas aritméticas más rápidas, como sería el producto de matrices. Se ha de recordar, desde luego, que los cálculos aritméticos en tiempo real para crear imágenes visuales sucesivas no constituyen necesariamente el mejor enfoque. A menudo es preferible calcular previamente los datos a partir de los cuales se construirán las figuras, antes de comenzar la secuencia del movimiento. Aun así, la técnica en tiempo real encaja con nuestros propósitos mejor que otra.

Las variables del BASIC se almacenan en la memoria encima del programa en BASIC. Estas variables se contienen en una tabla de variables, a cuyo inicio apunta el contenido de las posiciones 45 y 46 (decimal). Como todo puntero del Commodore 64, ambos están almacenados en la forma *lo-hi*. Por ello la dirección de inicio de la tabla de variables queda apuntada por la fórmula:

```
PEEK(45)+256*PEEK(46)
```

En el siguiente cuadro damos los distintos punteros asociados con las variables y su contenido normal:

Puntero	Función	Contenido normal
43/44	Inicio del BASIC	2049
45/46	Inicio de las variables	Depende de la longitud del prog.
47/48	Inicio de las tablas	Depende del núm. de variables
49/50	Fin de las tablas+1	Depende del núm./tamaño tablas
51/52	Vars. serie act. parte inf.	Depende del núm./long. variables
55/56	Parte sup. de la mem.	40960

Obsérvese que las variables dinámicas en serie se disponen a partir de la parte superior de la memoria según se van definiendo. Pero las tablas se almacenan por encima de las demás variables en la tabla de variables, y cuando en el curso de la ejecución del programa se encuentra una nueva variable, el OS desplaza toda el área de almacenamiento de tablas hacia arriba en el número de bytes necesario para almacenar la nueva variable.

El almacenamiento de variables en serie es por fuerza más complejo que el del resto de las variables. Las variables enteras (no ligadas en una tabla) y las variables de coma flotante sólo necesitan 7 bytes cada una, pero una variable en serie puede necesitar hasta 255 bytes. Para eludir esta complicación, el OS sólo almacena la longitud de la variable y un puntero que apunte a su dirección de inicio dentro de la tabla de variables. Si se define una variable literal en un programa BASIC (p. ej., A$="AMOR"), ésta apuntará al primer byte de la variable alfanumérica dentro del área para el programa BASIC. Tal variable se denomina *estática*. En el momento en que el programa altera su valor pasa a llamarse *dinámica*. Los valores de las variables alfanuméricas dinámicas se construyen a partir de la parte superior de la memoria cambiándose el valor del puntero en la tabla de variables. De esta manera toda entrada se mantiene en 7 bytes constantes.

En una tabla de variables podemos encontrar fácilmente la dirección de una variable desde el BASIC mediante un ligero trabajo de rastreo. El quid de la cuestión radica en que cada vez que el intérprete BASIC llame a una variable, su dirección vendrá apuntada por un puntero de la página cero situado en las direcciones decimales 71 y 72, que puede ser examinado para descubrir la dirección buscada. Pero hemos de apresurarnos en recabar esta información dado que la posición 71 también la emplea el sistema operativo cuando debe evaluar expresiones numéricas.

El programa que sigue ilustra el formato de variables ordinarias según son almacenadas en la memoria. La primera rutina recupera una variable alfanumérica de la memoria. Aquí empleamos la técnica que acabamos de describir para inspeccionar el contenido de las posiciones 71 y 72, almacenando inmediatamente estos valores en dos posiciones de reserva dentro del buffer para cassette. Serán empleadas posteriormente para calcular la dirección de la variable en serie y recuperarla.

```
1000 REM**BUSQUEDA DE VAR SERIE EN MEMORIA**
1010 X$="ABCDEF"
1020 REM**HACE X$ VARIABLE ACTUAL**
1030 X$=X$+" "
1040 REM**GUARDA PUNTERO DE TABLA VAR**
1050 POKE828,PEEK(71):POKE829,PEEK(72)
1060 REM**DIRECCION EN TABLA VAR**
1070 ADR=PEEK(828)+256*PEEK(829)
1080 REM**MIRA ENTRADA EN TABLA VAR**
1090 LS=PEEK(ADR):REM LONGITUD DE VAR SERIE
1100 SA=PEEK(ADR+1)+256*PEEK(ADR+2)
1110 REM SA ES LA DIR INICIO DE VAR SERIE
1120 REM**AHORA LEE LA VAR SERIE**
1130 FORI=SATOSA+LS
1140 VAR$=VAR$+CHR$(PEEK(I))
1150 NEXT
1160 PRINTVAR$
```

Puede que se le ocurra pensar que si empleamos variables enteras (indicadas con %, como X%) ahorraremos memoria y aceleraremos las operaciones aritméticas. Pero no es así en el Commodore 64. Cuando esta máquina tiene que realizar operaciones aritméticas con números enteros, los convierte a coma flotante y llama a las rutinas de coma flotante. Por eso, aunque las variables enteras se pue-



den almacenar en sólo dos bytes, se les reserva siete bytes de memoria si no están almacenadas en una tabla. Estos bytes extra serán ignorados en el curso del proceso.

La siguiente rutina localiza en la memoria una variable entera.

```
1000 REM**LOCALIZACION EN MEM. DE VAR ENTERA**
1010 X%=3456
1020 REM**HACE X% VARIABLE ACTUAL**
1030 X%=X%
1040 REM**GUARDA PUNTERO DE TABLA VAR**
1050 POKE828,PEEK(71):POKE829,PEEK(72)
1060 REM**DIRECCION EN TABLA VAR**
1070 ADR=PEEK(828)+256*PEEK(829)
1080 REM**MIRA ENTRADA EN TABLA VAR**
1090 LO=PEEK(ADR+1):HI=PEEK(ADR)
1100 REM**CALCULA RESULTADO**
1110 SIGNBIT=(HIAND128)/128
1120 VAR=LO+256*(HIAND127)-32768*SIGNBIT
1130 PRINT VAR
```

Esta misma técnica es válida para una variable de coma flotante. Hay, sin embargo, un método más económico. Se basa en que al servirnos de DEF FN para definir una función de la variable X, se emplea X (sin que cambie su valor) siempre que se llama a FN.

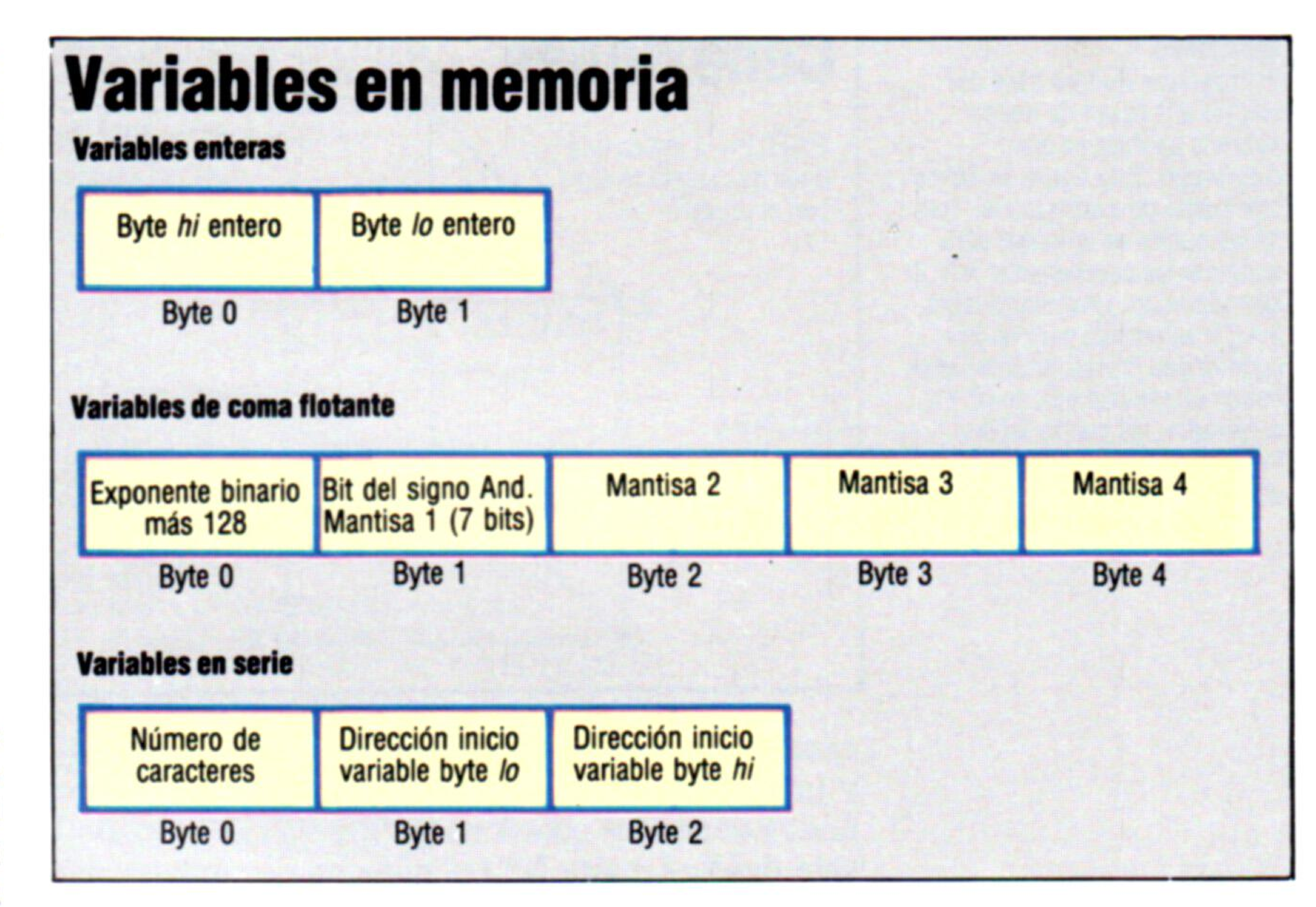
Este otro programa emplea DEF FN para calcular la dirección de la variable BASIC actualmente contenida en las posiciones 71 y 72. Como X sirve de variable de la función, estamos seguros de generar la dirección que pertenece al primer byte de X contenido en la tabla de variables del BASIC. Después se llama a FN para asignar esta dirección a la variable ADD. Obsérvese que el poner a cero (u otro valor) por medio de la instrucción FN no cambia el valor de X contenido en la tabla de variables, ni la dirección calculada por la función.

```
1000 REM**PARA HALLAR UN VAR DE CF EN MEMORIA**
1010 DEF FNADR(X)=PEEK(71)+256*PEEK(72)
1020 ADD=FNADR(0):REM DA SIEMPRE LA DIRECCION DE X
1030 X=-3.14159
1040 REM**CONVERSION BIN A DECIMAL**
1050 POWERTWO=2^(PEEK(ADD)-129)
1060 SIGN=(-1)^((PEEK(ADD+1)AND128)/128)
1070 REM*LA FRACCION ES DE 31 BITS*
1080 D1=PEEK(ADD+1)AND127:REM 7 BITS
1090 D2=PEEK(ADD+2):REM 8 BITS
1100 D3=PEEK(ADD+3):D4=PEEK(ADD+4)
1110 REM**GLUP**
1120 FRACT=2^(-7)*D1+2^(-15)*D2+2^(-23)*D3+2^
     (-31)*D4
1130 MANT=1+FRACT
1140 VAR=SIGN*POWERTWO*MANT
1150 PRINTVAR
```

Tablas en la memoria

La ejecución de la instrucción DIM hace que se reserve espacio de memoria para una tabla. Se compone de un byte de cabecera más todos los demás bytes necesarios para almacenar el elemento. El formato de los elementos almacenados en una tabla difiere según el tipo de variable que acabamos de ver. El cuadro adjunto resume estas variantes. Hay que comprender bien estas diferencias si se desea acceder a los elementos de una tabla desde código máquina.

Pero antes veamos algunos aspectos de las operaciones aritméticas con coma flotante. Cuando el intérprete del BASIC está realizando algún cálculo en coma flotante almacena los resultados intermedios en dos *acumuladores de coma flotante*. Se los denomina, respectivamente, FAC y ARG, y el formato que se emplea es el mismo que para el almacenamiento de variables en la memoria. FAC se encuentra en las direcciones $61 hasta la $65 (97 y 101 en decimal) y ARG ocupa de la $69 a la $6D (105 y 109). Para mayor sencillez continuaremos usando las rutinas del intérprete que sólo intercambian números entre FAC y la memoria.



Tipos de variables
Existen tres tipos de variables de tablas en el Commodore 64, cada una con un formato distinto en memoria. Las variables enteras se guardan en dos bytes como números en complemento a dos. Los números en coma flotante necesitan cinco bytes para guardar la mantisa, un bit de signo y el exponente. Las variables alfanuméricas se guardan en un área diferente de la memoria, tomando un byte por cada carácter de la serie. Sin embargo este dato no se guarda en la tabla; en su lugar, se usan tres bytes para dar el número de caracteres de la serie y la dirección de 16 bits que apunta al inicio del área de datos de la serie

Las rutinas del intérprete con las que trabajaremos en este apartado son:

- MOVFM (llamada en dirección $BBA2):
Esta rutina sirve para cargar el contenido de FAC desde una variable en coma flotante a la memoria. Se representa simbólicamente así: F←M. Para llamarla se cargará el acumulador con el byte *lo* de la dirección de inicio de la variable en memoria, y el registro Y con el byte *hi*.

- MOVMF (llamada en dirección $BBD4):
Esta rutina coloca el contenido de FAC en siete bytes de la memoria: M←F. Para llamarla se cargará el registro X con el byte *lo* y el registro Y con el byte *hi* del byte de inicio del destino de la variable en memoria.

- FMULT (llamada en dirección $BA28):
Es la rutina de la multiplicación, por la cual se multiplica el contenido de FAC por el valor de una segunda variable en memoria, y se almacena el resultado en FAC. La primera variable se coloca en FAC por medio de MOVFM y se apunta a la segunda variable cargando el acumulador con el byte *lo* y el registro Y con el byte *hi* del byte de inicio, antes de llamar a esta rutina. Finalmente, si es necesario, podemos devolver el resultado a la memoria empleando MOVMF.

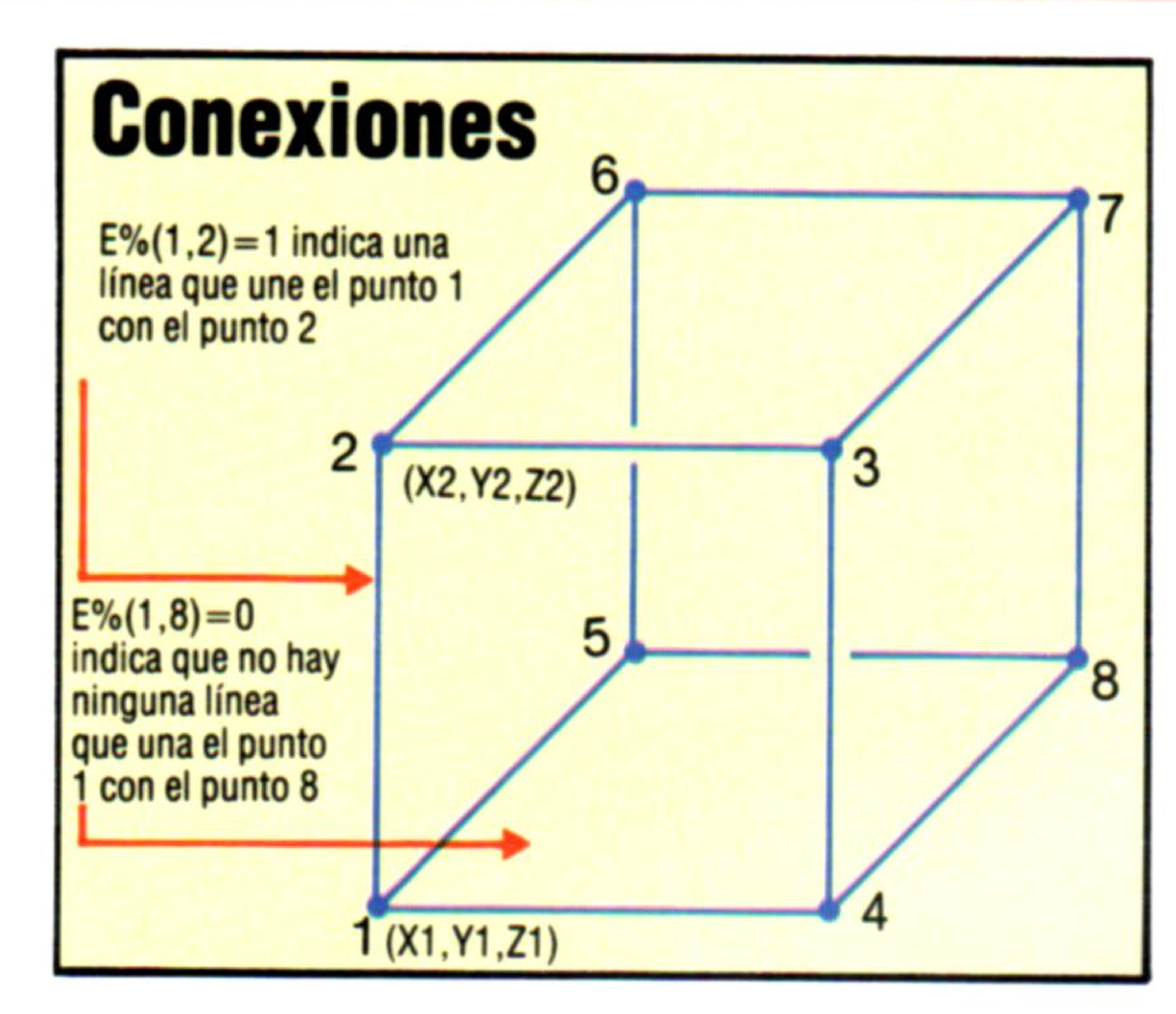
- FADD (llamada en dirección $B867):
Esta rutina de suma realiza esto: FAC=MEM−FAC. Para llamarla, hay que cargar el acumulador con el byte *lo* de MEM y el registro Y con el byte *hi* de MEM.

- FSUB (llamada en dirección $B850):
Esta rutina de resta realiza esto: FAC=MEM−FAC. Para llamarla cargamos el acumulador con el byte *lo* de MEM y colocamos el byte *hi* de MEM en el registro Y.

Éstas son las rutinas de intérprete que incorporaremos en la primera fase de nuestro programa de gráficos. Examinemos otros componentes de dicho programa.

La idea subyacente de este proyecto de gráficos es que la figura reticular de líneas puede especifi-

Bien atado
El programa *Rotsub* hace dar vueltas a la figura de líneas sobre la pantalla en alta resolución. Esta figura se define por medio de cuatro tablas, tres de las cuales se emplean para contener las coordenadas X,Y,Z de cada nodo. Una cuarta tabla, E%(,), se emplea para definir cuándo dos nodos determinados deben unirse con una línea. En el ejemplo, las cuatro tablas están inicializadas para producir una figura en forma de cubo



carse por medio de un número de puntos (o nodos) y una matriz de conexiones de los extremos. Los nodos tienen las coordenadas X(I),Y(I),Z(I), donde I vale desde 1 hasta NP (el número de puntos). La matriz de conexiones de esquinas es E%(I,J), donde I y J van desde 1 hasta NP. Si el punto I se conecta con el punto J,E%(I,J) valdrá uno. De lo contrario, será igual a cero. Este procedimiento no es, desde luego, el más económico en memoria, dado que empleamos dos bytes cuando bastaría con uno. Pero se facilita al usuario la tarea de especificar los puntos que han de conectarse. En aplicaciones prácticas, NP tampoco será muy grande.

Para comunicarnos desde el BASIC con la rutina de rotación en código máquina, colocaremos (POKE) las direcciones de base de las tablas comenzando por el elemento 1. Así, las direcciones X(1),Y(1),Z(1) y E%(1,1) no es necesario hallarlas. Además, suministraremos al código máquina el número de puntos, NP, y el coseno (COS) y seno (SIN) del ángulo de rotación deseado alrededor del eje Z.

Todo proyecto ambicioso en código máquina debe proceder por pequeños pasos si no se desea caer en errores desastrosos más adelante. Por ello hemos planificado el proyecto en tres fases. En la primera definimos los algoritmos y escribimos el texto del programa en BASIC. Esto se realiza en el programa *Cubo giratorio en* BASIC (página contigua), que hace dar vueltas a un cubo de tres dimensiones alrededor del eje Z y proyecta el resultado en el plano X,Y.

Nuestro otro objetivo es convertir el listado BASIC en código máquina. Pero comenzamos con una tarea menos ambiciosa consistente en dar la subrutina de la línea 1800 en su versión de código máquina. Esto es lo que hace el programa *I-Rotsub*, dando además un programa de comprobación de esta rutina.

En el próximo capítulo completaremos el proyecto y añadiremos el resto de la codificación al listado assembly, así como daremos un "Cargador en BASIC" del programa completo.

Programa I-Rotsub 64

El siguiente listado en assembly será ensamblado y cargado y el código resultante cargado como I-ROT.HEX

```
; ++++++++++++++++++++++
;++         I-ROTSUB 64          ++
; ++++++++++++++++++++++
;++    EMPLEA TABLAS DEFINIDAS    ++
;++ DESDE BASIC SE DEBEN COLOCAR ++
;++    LAS DIRECCIONES PRIMERO    ++
; ++++++++++++++++++++++
;
;PLTSUB    = $C183
;XLO       = $C103
;XHI       = $C104
;YLO       = $C105
;
*          = $C544
;
;++++ VARIABLES ROTSUB ++++
;
; VARIABLES LLAMADAS DESDE BASIC
XBASLO     *=*+1 ; POKE50500,X(0)LO
XBASHI     *=*+1 ; POKE50501,X(0)HI
XBASLO     *=*+1 ; POKE50502,Y(0)LO
YBASHI     *=*+1 ; POKE50503,Y(0)HI
NP         *=*+1 ; POKE50504,NP
; VARIABLES EMPLEADAS POR C/M
XILO       *=*+1
XIHI       *=*+1
YILO       *=*+1
YIHI       *=*+1
CSLO       *=*+1
CSHI       *=*+1
SNLO       *=*+1
SNHI       *=*+1
MEM1       *=*+5 ; VAR COMA FLOTANTE
MEM2       *=*+5 ; VAR COMA FLOTANTE
FAC        =$0061
ARG        =$0069
;
; LLAMADAS ARITMETICAS AL INTERPRETE
;
FMULT      =$BA28   ; FAC=FAC*ARG
FADDT      =$B86A   ; FAC=FAC+ARG
FSUB       =$B850   ; FAC=MEM-FAC
FADD       =$B867   ; FAC=FAC+MEM
MOVFM      =$BBA2   ; FAC=MEMORY
MOVMF      =$BBD4   ; MEMORY=FAC
;
;++++ GUARDA LOS REGISTROS
;
       PHA
       TXA
       PHA
       TYA
       PHA
;
;++++ INICIALIZA VARIABLES ++++
;
       LDA XBASLO
       STA XILO
       LDA XBASHI
       STA XIHI
       LDA YBASLO
       STA YILO
       LDA YBASHI
       STA YIHI
;
;++++ REALIZA MEM1=X(I)*CS-Y(I)*SN
;
START
       LDA XILO
       LDY XIHI
       JSR MOVFM     ; FAC=X(I)
       LDA CSLO
       LDY CSHI
       JSR FMULT     ; FAC=X(I)*CS
       LDX #<MEM1
       LDY #>MEM1
       JSR MOVMF     ; MEM1=X(I)*CS
       LDA YILO
       LDY YIHI
       JSR MOVFM     ; FAC=Y(I)
       LDA SNLO
       LDY SNHI
       JSR FMULT     ; FAC=Y(I)*SN
       LDA #<MEM1
       LDY #>MEM1
       JSR FSUB      ; FAC=MEM1-FAC
       LDX #<MEM1
       LDY #>MEM1
       JSR MOVMF     ; MEM1=FAC
;
;++++ REALIZA MEM2=Y(I)*CN+X(I)*SN
;
       LDA YILO
       LDY YIHI
       JSR MOVFM     ; FAC=Y(I)
       LDA CSLO
       LDY CSHI
       JSR FMULT     ; FAC=Y(I)*CS
       LDX #<MEM2
       LDY #>MEM2
       JSR MOVMF     ; MEM2=Y(I)*CS
       LDA XILO
       LDY XIHI
       JSR MOVFM     ; FAC=X(I)
       LDA SNLO
       LDY SNHI
       JSR FMULT     ; FAC=X(I)*SN
       LDA #<MEM2
       LDY #>MEM2
       JSR FADD      ; FAC=MEM2+FAC
       LDX #<MEM2
       LDY #>MEM2
       JSR MOVMF     ; MEM2=FAC
;
;++++ REALIZA X(I)=MEM1:Y(I)=MEM2
;
       LDA #<MEM1
       LDY #>MEM1
       JSR MOVFM     ; FAC=MEM1
       LDX XILO
       LDY XIHI
       JSR MOVMF     ; X(I)=FAC
       LDA #<MEM2
       LDY #>MEM2
       JSR MOVFM     ; FAC=MEM2
       LDX YILO
       LDY YIHI
       JSR MOVMF     ; Y(I)=FAC
;
;++++ BUCLE COMPROBACION FIN
;
       DEC NP
       BEQ EXIT
;
;++++ INCREMENTA PUNTEROS TABLA
;
       LDA #$05
       CLC
       ADC XILO
       STA XILO
       BCC XNOHI
       INC XIHI
XNOHI
       LDA #$05
       CLC
       ADC YILO
       STA YILO
       BCC YNOHI
       INC YIHI
YNOHI
       JMP START
;
;++++ SACA REGISTROS DE LA PILA ++++
;
EXIT
       PLA
       TAY
       PLA
       TAX
       PLA
;
       RTS
.END
```

Cubo giratorio en BASIC

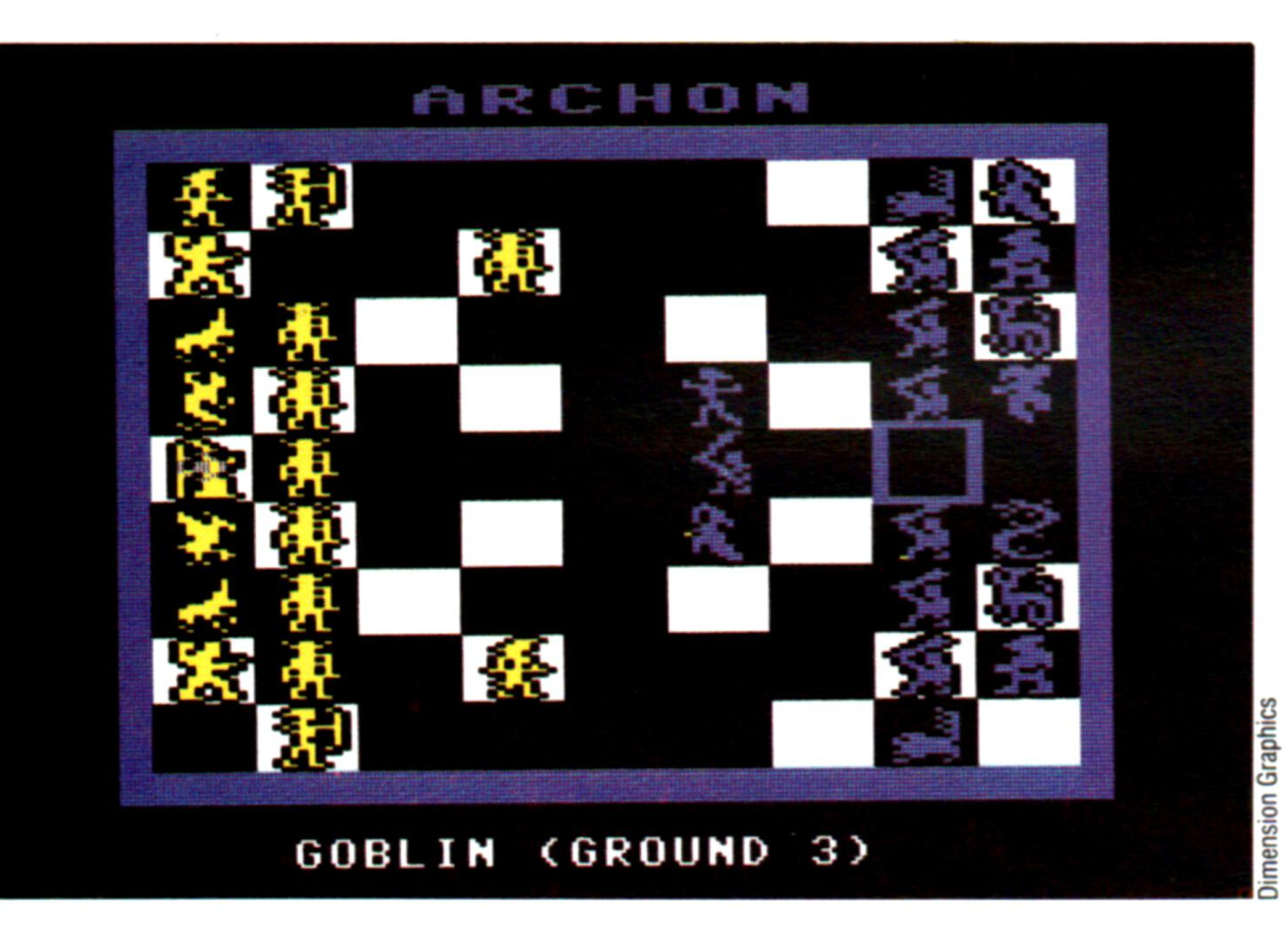
```
1000 REM**CUBO GIRATORIO EN BASIC**
1010 IFA=0THENA=1:LOAD"PLOTSUB.HEX",8,1
1020 IFA=1THENA=2:LOAD"LINESUB.HEX",8,1
1030 REM**DIMENSION TABLAS**
1040 NP=8:REM NUMERO DE PUNTOS
1050 DIM X(NP),Y(NP),Z(NP)
1060 DIM ED(NP,NP):REM CONEXION ESQUINAS
1070 REM**INICIALIZACION TABLAS
1080 REM--DATOS COORDENADAS CUBO
1090 DATA  75,  75,  75:REM  --------- /1
1100 DATA -75,  75,  75:REM  CUATRO PTS /2
1110 DATA -75, -75,  75:REM  SUPERIORES /3
1120 DATA  75, -75,  75:REM  --------- /4
1130 DATA  75,  75, -75:REM  --------- /5
1140 DATA -75,  75, -75:REM  CUATRO PTS /6
1150 DATA -75, -75, -75:REM  INFERIORES /7
1160 DATA  75, -75, -75:REM  --------- /8
1170 REM**CUBO GIRA PI/4 SOBRE EJE *
1180 FORI=1 TONP
1190 READX(I),Y(I),Z(I)
1200 Y(I)=Y(I)*COS(π/4)-Z(I)*SIN(π/4)
1210 Z(I)=Z(I)*COS(π/4)+Y(I)*SIN(π/4)
1220 NEXT
1230 REM**CUBO GIRA PI/4 SOBRE EJE Z
1240 FORI=1 TONP
1250 X(I)=X(I)*COS(π/4)-Y(I)*SIN(π/4)
1260 Y(I)=Y(I)*COS(π/4)+X(I)*SIN(π/4)
1270 NEXT
1280 REM--DATOS CONEXION ESQUINAS--
1290 E(1,2)=1:REM 1 CONECTA CON 2
1300 E(2,3)=1:E(3,4)=1:E(4,1)=1
1310 E(5,6)=1:REM CUADRO DE BASE
1320 E(6,7)=1:E(7,8)=1:E(8,5)=5
1330 E(5,1)=1:REM ESQUINAS DE ARRIBA ABAJO
1340 E(6,2)=1:E(7,3)=1:E(8,4)=1
1350 REM**SIMETRIA E(I,J)**
1360 FORI=1 TONP:FORJ=1 TONP
1370 IFE(I,J)=1 THENE(J,I)=1
1380 NEXT:NEXT
1390 REM######################
1400 REM##TRAZADO CUBO GIRATORIO##
1410 SA=2*π/45
1420 FOR A=π/4 TO π/4+2*π STEP SA
1430 GOSUB1800:REM GIRA POR SA
1440 GOSUB1590:REM INIC/BORR PANTALLA
1450 REM--TRAZADO CUBO--
1460 FORI=1 TONP
1470 FORJ=1 TOI
1480 IF E(I,J)=0THEN1510:REM NO UNIDO
1490 GOSUB1630:REM CALCULA PROYECCION
1500 GOSUB1670:REM UNE PUNTOS
1510 NEXT:NEXT
1520 REM-------------
1530 NEXT A:REM ANGULO SIGUIENTE
1540 REM######################
1550 REM**ESPERA**
1560 GETA$:IFA$=""THEN1560
1570 GOSUB1760:REM RESTAURA PANTALLA
1580 END
1590 REM**ESTABLECE ALT. RES**
1600 POKE49408,1:POKE49409,1
1610 POKE49410,1:SYS49422
1620 RETURN
1630 REM**CALCULA PROYECCION EN ALT. RES.**
1640 X1%=X(I)+159:Y1%=199-(Z(I)+100)
1650 X2%=X(J)+159:Y2%=199-(ZJ)+100)
1660 RETURN
1670 REM**LINESUB**
1680 IF(X1%=X2%)AND(Y1%=Y2%)THENRETURN
1690 MHI=INT(X1%256):MLO=X1%-256*MHI
1700 NHI=INT(X2%ı256):NLO=X2%-256*NHI
1710 POKE49920,MLO:POKE49921,MHI
1720 POKE49922,NLO:POKE 49923,NHI
1730 POKE49924,Y1%:POKE49925,Y2%
1740 SYS49934:REM LINESUB
1750 RETURN
1760 REM**RESTAURA PANTALLA**
1770 POKE49408,0:SYS49422
1780 PRINTCHR$(147)
1790 RETURN
1800 REM**CUBO GIRA SOBRE EJE Z/SA
1810 FORI=1 TONP
1820 X(I)=X(I)*COS(SA)-Y(I)*SIN(SA)
1830 Y(I)=Y(I)*COS(SA)+X(I)*SIN(SA)
1840 NEXT
1850 RETURN
```

Programa prueba I-Rotsub

Debe entrarse y guardarse con el nombre PRUEBA I-ROT. No pueden definirse variables mientras se ejecuta el código entre 1880 y 2000 y se llama al SYS50523. De lo contrario la dirección de base de las tablas se pasaría incorrectamente al cód. máq. y el programa se arruinaría

```
1000 REM** PRUEBA I-ROT **
1010 IFA=0THENA=1:LOAD"PLOTSUB.HEX",8,1
1020 IFA=1THENA=2:LOAD"LINESUB.HEX",8,1
1030 IFA=2THENA=3:LOAD"I-ROT.HEX",8,1
1040 REM**DIMENSION TABLAS**
1050 NP=8:REM NUMERO DE PUNTOS
1060 DIM X(NP),Y(N),Z(NP)
1070 DIM ED(NP,NP):REM CONEXION ESQUINAS
1080 REM**INICIALIZACION TABLAS**
1090 REM--DATOS COORDENADAS CUBO
1100 DATA  75,  75,  75:REM  ---------- /1
1110 DATA -75,  75,  75:REM  CUATRO PTS /2
1120 DATA -75, -75,  75:REM  SUPERIORES /3
1130 DATA  75, -75,  75:REM  ---------- /4
1140 DATA  75,  75, -75:REM  ---------- /5
1150 DATA -75,  75, -75:REM  CUATRO PTS /6
1160 DATA -75, -75, -75:REM  INFERIORES /7
1170 DATA  75, -75, -75:REM  ---------- /8
1180 REM**GIRO SOBRE EJE X EN PI/4
1190 FORI=1 TONP
1200 READX(I),Y(I),Z(I)
1210 Y(I)=Y(I)*COS(π/4)-Z(I)*SIN(π/4)
1220 Z(I)=Z(I)*COS(π/4)+Y(I)*SIN(π/4)
1230 NEXT
1240 REM**GIRO SOBRE EJE Z EN PI/4
1250 FORI=1 TONP
1260 X(I)=X(I)*COS(π/4)-Y(I)*SIN(π/4)
1270 Y(I)=Y(I)*COS(π/4)+X(I)*SIN(π/4)
1280 NEXT
1290 REM--DATOS CONEXION ESQUINAS--
1300 E(1,2)=1:REM 1 CONECTADO A 2
1310 E(2,3)=1:E(3,4)=1:E(4,1)=1
1320 E(5,6)=1:REM CUADRADO INFERIOR
1330 E(6,7)=1:E(7,8)=1:E(8,5)=5
1340 E(5,1)=1:REM ESQUINAS DE ARRIBA ABAJO
1350 E(6,2)=1:E(7,3)=1:E(8,4)=1
1360 REM**SIMETRIA E(I,J)**
1370 FORI=1 TONP:FORJ=1 TONP
1380 IFE(I,J)=1THENE(J,I)=1
1390 NEXT:NEXT
1400 REM######################
1410 REM##TRAZADO CUBO GIRATORIO##
1420 SA=2*π/45:CS=COS(SA):SN=SIN(SA)
1430 FOR A=0 TO 2*π STEP SA
1440 GOSUB1870:REM GIRO POR SA
1450 GOSUB1600:REM INIC/BORRADO PANTALLA
1460 REM--TRAZADO CUBO--
1470 FORI=1 TONP
1480 FORJ=1TOI
1490 IF E(I,J)=0THEN1520:REM NO UNIDO
1500 GOSUB1640:REM CALCULO PROYECCION
1510 GOSUB1680:REM UNION PUNTOS
1520 NEXT:NEXT
1530 REM-------------
1540 NEXT A:REM ANGULO SIGUIENTE
1550 REM######################
1560 REM**ESPERA**
1570 GETA$:IFA$=""THEN1570
1580 GOSUB1770:REM RESTAURA PANTALLA
1590 END
1600 REM**ESTABLECE ALT. RES.**
1610 POKE49408,1:POKE49409,1
1620 POKE49410,1:SYS49422
1630 RETURN
1640 REM**CALCULA PROYECCION EN ALT. RES.**
1650 X1%=X(I)+159:Y1%=199-(Z(I)+100)
1660 X2%=X(J)+159:Y2%=199-(Z(J)+100)
1670 RETURN
1680 REM**LINESUB**
1690 IF(X1%=X2%)AND(Y1%=Y2%)THENRETURN
1700 MHI=INT(X1%/256):MLO=X1%-256*MHI
1710 NHI=INT(X2%/256):NLO=X2%-256*NHI
1720 POKE49920,MLO:POKE49921,MHI
1730 POKE49922,NLO:POKE49923,NHI
1740 POKE49924,Y1%:POKE49925,Y2%
1750 SYS49934:REM LINESUB
1760 RETURN
1770 REM**RESTABLECE PANTALLA**
1780 POKE49408,0:SYS49422
1790 PRINTCHR$(147)
1800 RETURN
1810 REM**GIRO CUBO SOBRE EJE Z/SA
1820 FORI=1 TONP
1830 X(I)=X(I)*CS-Y(I)*SN
1840 Y(I)=Y(I)*CS+X(I)*SN
1850 NEXT
1860 RETURN
1870 REM**GIRO SOBRE EJE Z/SA
1880 X(1)=X(1):REM VAR X(1) ACTUAL
1890 POKE50500,PEEK(71):REM BYTE LO DE X(1)
1900 POKE50501,PEEK(72):REM BYTE HI DE X(1)
1910 Y(1)=Y(1):REM VAR Y(1)ACTUAL
1920 POKE50502,PEEK(71):REM BYTE LO DE Y(1)
1930 POKE50503,PEEK(72):REM BYTE HI DE Y(1)
1940 POKE50504,NP:REM NUMERO DE PUNTOS
1950 CS=CS:REM HACE A CS VAR ACTUAL
1960 POKE50509,PEEK(71)
1970 POKE50510,PEEK(72)
1980 SN=SN:REM HACE A SN VAR ACTUAL
1990 POKE50511,PEEK(71)
2000 POKE50512,PEEK(72)
2010 SYS50523
2020 RETURN
```

Luz y sombra



Dimension Graphics

En guardia
La forma más apropiada de definir *Archon* es decir que se trata de un juego de ajedrez animado. Cuando se le pide a una pieza que se desplace a otro cuadrado del tablero, marchará, se escabullirá o volará, según el tipo de personaje que represente. Una vez allí, si la casilla está ocupada por una pieza del bando oponente, la pantalla pasará a un primer plano del cuadrado y comenzará la batalla. El resultado de ésta se decide en función de la destreza del jugador y de la fortaleza relativa de las piezas contrarias

El juego "Archon" combina, con gran originalidad, elementos propios del ajedrez con gráficos de movimiento rápido

Archon es un juego que puede resultarle fascinante al usuario aficionado al ajedrez y que, al mismo tiempo, puede satisfacer al más apasionado entusiasta de los juegos recreativos. La eterna batalla entre fuerzas de la Luz y la Oscuridad constituye el nudo de la trama. El juego comienza con una visualización de un tablero de estrategia en el cual los dos bandos contendientes están dispuestos en filas y columnas, de forma muy similar a las piezas del ajedrez. Las piezas poseen nombre tales como "fénix" y "caballero" en el lado de Luz, mientras que "espíritus", "gnomos" y "dragones" distinguen a las fuerzas de la Oscuridad. Los iconos que representan las piezas de ambos bandos poseen diferentes fuerzas y capacidades de movimiento. Algunas piezas, por ejemplo, pueden "volar", lo que significa que pueden saltar por encima de las piezas que tienen por delante, mientras que otras tienen sólo un movimiento a nivel de "tierra".

El tablero propiamente dicho está dividido en una cuadrícula de nueve por nueve. A primera vista, parece muy similar a un tablero de ajedrez, con muchos de los cuadrados coloreados alternadamente en blanco y negro. Sin embargo, otras casillas cambian de blanco a negro y otra vez a blanco a medida que va transcurriendo el juego. El motivo de ello es que las fuerzas de la Luz son más fuertes en los cuadrados blancos y las fuerzas de la Oscuridad son más fuertes en los negros.

El objetivo de *Archon* consiste en ocupar los cinco "puntos de poder"; cuatro están situados en cruz sobre los bordes del tablero, con el punto crucial en el cuadrado del centro. Cuando comienza el juego, la estrategia de apertura determina que usted mueva sus iconos desde su color opuesto, donde son vulnerables al ataque, a colores en los cuales son más fuertes. Al hacer esto, también puede abrir la fila trasera de modo que las piezas de "tierra" queden libres para moverse, permitiéndole construir una barrera eficaz para impedir que las fuerzas de tierra contrarias desbaraten la defensa.

En este punto, el juego parece tener transferidos al ordenador la mayoría de los elementos del ajedrez. No obstante, la verdadera diferencia del juego se hace evidente al intentar ocupar un cuadrado que ya esté en poder de un icono contrario. Cuando se mueve un icono a un cuadrado ocupado por el enemigo, mediante la palanca de mando, la pantalla revela un primer plano del cuadrado, en vez de capturar automáticamente la pieza. Las piezas del enemigo están posicionadas a los lados del cuadrado, en el cual hay un barra que representa la fuerza del icono, y comienza una contienda estilo recreativo. Cuando la barra es golpeada por la pieza contraria, esta fuerza disminuirá, y cuando desaparezca por completo, el enemigo gana y ocupa el cuadrado. Dado que las fortalezas y los métodos de ataque difieren radicalmente de un icono a otro. Algunas de estas batallas están más niveladas que otras. Por ejemplo, un dragón puede lanzar su llama a través del cuadrado a un contrincante, mientras que un caballero ha de estar muy cerca de un enemigo para poder hacer uso de su espada. Las batallas se complican aún más en función de barreras diseminadas a través del cuadrado, que cambian a medida que transcurre la acción.

Cada bando cuenta con un icono que puede realizar un encantamiento: un hechicero para la Luz y una hechicera para la Oscuridad. Los encantamientos son idénticos, pero el icono sólo los puede utilizar una vez. Los hechizos incluyen poderes tales como "resucitar" una pieza en un cuadrado alejado de su capacidad de movimiento, o invocar a una criatura conocida como un "elemental". A éstas se las puede convocar para que ataquen a una pieza enemiga y son útiles cuando una pieza enemiga amenaza atacar a una de las suyas mucho más débil, o si usted desea debilitar a su oponente. Sin embargo, incluso si el elemental destruye al icono enemigo, al final de la batalla desaparece. Un elemental no puede ocupar un cuadrado.

Archon se puede jugar contra el ordenador o bien contra otra persona. Aunque el ordenador es sólo un estratega mediano, en la sección recreativa resulta ser un oponente formidable, y puede que en las primeras partidas el principiante se encuentre demasiado desbordado para ganar siquiera una sola batalla.

Archon: Para el Commodore 64
Editado por: Ariolasoft, Dept A.S. 72, Westfields Avenue, London, SW13 0AU, Gran Bretaña
Autores: Anne Westfall, Jon Freeman, Paul Reiche III
Palanca de mando: Necesaria
Formato: Cassette

Editorial Delta, S.A.



10087
9 788485 822836